Europäisches Patentamt
European Patent Office
Office européen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

02079196.8

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 02079196.8
Demande no:

Anmeldetag:
Date of filing: 08.10.02
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

Koninklijke Philips Electronics N.V.
Groenewoudseweg 1
5621 BA Eindhoven
PAYS-BAS

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Integrated circuit and method of operating

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F13/00

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR IE IT LI LU MC NL PT SE SK TR

08.10.2002

Systems on silicon show a continuous increase in complexity due to the ever increasing need for implementing new features and improvements of existing functions. This is enabled by the increasing density with which components can be integrated on an integrated circuit. At the same time the clock speed at which circuits are operated tends to increase too. The higher clock speed in combination with the increased density of components has reduced the area which can operate synchronously within the same clock domain. This has created the need for a modular approach. According to such an approach the processing system comprises a plurality of relatively independent, complex modules. In conventional processing systems the systems modules usually communicate to each other via a bus. As the number of modules increases however, this way of communication is no longer practical for the following reasons. On the one hand the large number of modules forms a too high bus load. On the other hand the bus forms a communication bottleneck as it enables only one device to send data to the bus. A communication network forms an effective way to overcome these disadvantages. The communication network comprises a plurality of partly connected nodes. Messages from a module are redirected by the nodes to one or more other nodes. To that end the message comprises first information indicative for the location of the addressed module(s) within the network. The message may further include second information indicative for a particular location within the module, such as a memory, or a register address. The second information may invoke a particular response of the addressed module.

It is an object of the invention to provide an integrated circuit and a method according to the introductory paragraph, which provides the modules therein a relatively simple way of issuing messages.

In order to achieve said object the integrated circuit is characterized by the characterizing portion of claim 1.

In the integrated circuit according to the invention modules can issue messages in a simple way, by using a single address. This makes it possible for a module to perform a write action to a particular memory address without being aware of the destination which comprises said address is stored.
In this way the network appears to the model issuing the message as a bus. This makes it relatively simple to incorporate already existing modules designed for a bus like architecture in an integrated circuit according to the invention.

As such, processing systems are known, where a processor is coupled via a bus to various memories, which each are mapped onto a respective portion of the total address range. By way of example a ROM and a RAM may be mapped to a first and a second address range respectively. When the processor performs a read instruction, the address in the instruction defines at the same time which memory is selected to read the data from.

In such known processing systems each of the various modules, such as memories are directly coupled to the bus. In the integrated circuit according to the invention, selecting one of the modules implies that the one or other memories are set in a state wherein they do not interfere with the bus traffic. Apart from the memory that is addressed no other module is required to perform an action (in fact, they don't have to and don't need to know that another module is active – i.e. they don't have to be 'set in a state'), or 2) that multiple concurrent and/or pipelined messages can be active simultaneously in the network as a whole. In an integrated circuit according to the invention however, information issued by the active module is transferred as a message via one or more nodes of the network. As a consequence it follows a different route through the network depending on the address. This route is scheduled by the network.

Examples of the two pieces of information that are arranged as a single address are:
Single logical memory space/map/range mapped to multiple distributed memories each with their own physical memory ranges.
Virtual memory space mapped to a single logical memory space (distributed or not).
Multiple memory spaces/maps/ranges mapped to multiple distributed memories. For 2) and 3) two translations may take place (vm -> logical -> physical, and multiple -> single -> physical).

The integrated circuit of claim 3 and the method of claim 4 provide another way of improving data transfer in an integrated circuit comprising a plurality of modules connected by a network.
Theoretically a transaction could comprise any number of outgoing and/or return messages. In practice however a transaction is made up of one or two outgoing messages (from the first to the second module), and zero, one, or two return messages (from the second to the first module). By managing the outgoing messages in a way different from the return messages the overall efficiency of the network and therewith the integrated circuit comprising the network is improved. This is further illustrated with the following embodiments.

With reference to claim 5 it is remarked that GT connections can overbook resources in some cases. For example, when an ANIP opens a GT read connection, it must reserve slots for the read command messages, and for the read data messages. The ratio between the two can be very large (e.g., 1:100), which leads either to large slot tables, or bandwidth being wasted for the read command messages. In order to prevent as much as possible that a reservation for guaranteed traffic would impede other transactions the bandwidth which can be reserved should be restricted. On the other hand the best effort traffic may use any resources which are currently available. As a consequence guaranteed traffic has bounded but on average higher latency than best-effort traffic which has no fixed upper bound, but is (or should be) faster on average.
Based on this recognition it has been found that the overall quality of the network transport could be improved by **exploiting BE packets for read command messages, and GT packets for read data messages.** No guarantees can be offered in this case, but the overall throughput can be higher and more stable than in the case of using only BE packets.

08.10.2002

With reference to claim 6 it is remarked that preferably the outgoing transactions are handled in a locally ordered and the return transactions in a globally ordered transaction mode. The one or more adressed modules process the transactions in the order they have been issued, and the return part of the transactions are all delivered to the first module in the order in which it initiated the transactions. Even if ordered channels are used, the responses from different addressed modules (e.g., in a narrow cast connection) must be sorted at the first module. This kind of ordering conforms with AMBA

To implement global ordering, transactions that are delivered to different second modules (also referred to as slave) must be ordered exactly as they were sent by the first module (also referred to as master). This means that the network should either have a global time indicator, and use e.g. deadline-based scheduling in the network while in addition assumption on the consumption time of the second models must be available. An alternatively way to introduce global ordering is to introduce explicit dependencies between transactions. The latter can be done by using acknowledged/tagged transactions, where proof of delivery to the slave is sent back to the master using an acknowledgement message. This solution, however, introduces extra latency because transactions are sequentialised with a round-trip delay/latency per transaction, (send a message, wait for the acknowledgement, send next message, wait for next acknowledgement, etc.). By requiring only a local ordering for the delivery of the outgoing transactions, the slaves, provided that they are autonomous (which is usually the case) can execute messages independently.

With reference to claim 7 it is remarked that in this way buffer space is used in an efficient way. A particular example is an embodiment wherein a large buffer space is reserved for the buffer of the network interface coupled to an active module, such as a module isuing a read command, and a small buffer space is reserved for the buffer of the network interface coupled to a passive module, e.g. the one receiving the read message.

In other situations there may be different types of flow control (e.g. you never want to lose write commands, but don't mind losing read data). If a module can do both read and write commands, it may be important that write transactions always succeed (e.g. when writing to an interrupt controller), but that read transactions are not critical because they can be retried (so the CMD of the read transaction is dropped and the read never executed, or the RETDATA is dropped after the read has been executed. Another example is that if you know that writes always succeed if they are delivered, a flow-controlled connection is requested. Acknowledgements are not necessary in that case; Without flow control acknowledgements are compulsory, complicating the master and causing additional traffic.

In the integrated circuit according to the invention the decision to drop messages or not is not decided per transaction but for the outgoing and return parts of connection as a whole. For example all outgoing messages having the format reads+address or writes+address+data) may be guaranteed lossless, while for all return messages (whether read data, write acknowledgements) packets may be dropped.

A connection could be opened as follows:
connid = open (

08.10.2002

nofc/fc,
outgoing unordered/local/global,
outgoing buffer size,
return unordered/local/global,
return buffer size);
i.e. all outgoing messages have certain properties, and all return messages have certain properties.

With reference to claim 8 it is remarked that in a processing system with modules working asynchronously with respect to each other it is usual that a module receiving data issues an acknowledge signal to inform the issuing processor that it has received a message. In case that a message is multicast a plurality of said acknowledge signals is generated, which imposes a burden for the issuing processor. In the integrated circuit of the invention the first module receives only a single message, which reduces this burden. This measure is based on the insight that the network usually can relatively easily generate the single return message in response to the plurality of acknowledge messages of the second modules as a side effect of the functions already present in the network for other purposes.

With reference to claim 9: Depending on the situation the single return message can depend on the acknowled messages in various ways. The embodiment of claim 2 is favorable where the addressed second modules are memories, and the first module attempts to store data therein. In that case it is sufficient that only one copy of the data is really received and stored.

With reference to claim 10: In other situations it is compulsory that each of the addressed second modules has received the data. In the embodiment of claim 10 the single return message is not generated until this is the case.
Otherwise the returnn message could be combined as follows.
If each of the write transaction has been successfully executed by all slaves, all will return RETSTAT=RETOK, which can be combined by
the ANIP in a single message to be delivered to the master.

If the write transaction has been successfully executed only by some slaves, there will be a mix of RETSTATs (RETOK and RETERROR). They can either be combined into
(a) a single RETSTAT=RETERROR, to specify that an error occured, or
(b) a single RETSTAT, but a larger one, more descriptive, encoding where there have been errors. All RETSTATs can be bundled together in a single RETSTAT for the master, or <slave identifiers,error code> pairs can be bundled to form a single RETSTAT for the master.

If the connection has no flow control, messages can be dropped
at the PNIPs, resulting also in RETSTAT=RETLOST messages. Again, combinations as those above can be made.

With reference to claim 11: In this way it is guaranteed that the first module always receives a response to a transaction, even if the connection has no flow control (i.e. data may be dropped). This is done by only dropping data in the PNIP (the network interface coupled to the second, receiving module), and returning a FAIL/ERROR to the ANIP (The network interface coupled to the first module). This return status (RETSTAT) message will never be dropped because the ANIP that initiated the transaction will reserve space for return messages of every transaction that it initiates. This combination of reserving space and generating an error message whenever a message is dropped is a way to introduce flow control. Preferably the RETSTAT message is generated by the interface of the receiving module, although alternatively it could be generated at the intermediary network nodes too.
The method according to the invention guarantees transaction completion, i.e. it is always known whether an initiated transaction
(a) was delivered and executed successfully at the slave (RETSTAT=OK produced by the slave), or
(b) was never delivered at the slave (RETSTAT=REQLOST produced by the PNIP), or
(c) was delivered at the slave, but not successfully executed (RETSTAT=ERROR produced by the slave), or
(d) was delivered and executed successfully at the slave but the response message was dropped (RETSTAT=RETLOST produced by the ANIP).
This is achieved by either
(i) not dropping messages (flow-controlled connection), in this case RETSTAT is either OK or ERROR, or
(ii) by allowing messages to be dropped (on a connection without flow control), but generating a RESTAT (REQLOST or RETLOST) whenever the message is dropped, or a RETOK or RETERROR as usual when the message is not dropped.

It is essential however, never to drop RETSTATs, because this completes the transaction.This is realized in that a buffer for the RETSTAT is located at the master's ANIP. The latter reserves space for RETSTATs when initiating transactions, and bounds the number of outstanding transactions (for finite sized RETSTAT buffers).

The flow control on the outgoing and return connections is in principle independent. Thus, for outgoing flow control & return flow control, the RETSTAT message is according to a) or c) above
In case of outgoing flow control & no return flow control, the RETSTAT message is a) or c) or d) above.
In case of no outgoing flow control & return flow control, the RETSTAT message is a) or b) or c) above.

Other embodiments are such an integrated circuit wherein the return message is a message indicating whether the second module has received a message from the first module. In this embodiment the return message can be very compact, e.g.. one or two bits to indicate one of the four options described above.
Alternatively or in addition a return message comprises an identification of the message received by the second module.

Page: 7
1. I suggest "efficiency" instead of "performance", because performance is just one of the

factors. We may have the option to reduce the cost of the network (e.g., reduce buffer sizes), or increase the performance (e.g., by adding more connections for the same resources).
Page: 8

2. This is an example for the use of different properties for outgoing and return parts. However, more can be defined:

√ Acknowledged write transaction: write command + outgoing data use guaranteed throughput (mode one in your example), and acknowledgment uses best effort (mode two in your example)

Moreover, except time-related guarantees, there is also a distinction on the buffering in both you and the above example. For data messages there is potentially more buffering allocated than for commands and acknowledgments. Consequently, for a read transaction (your example) buffers for the return part would be larger than those for the outgoing part, For the acknowledged write (the example above), buffers for the outgoing part are larger, and those for acknowledgments are smaller.
Page: 8

3. It is indeed possible to allocate different bandwidths as you suggest. However, there are also limitations. We use a slot table, which contains a number of slots in a time window. Bandwidth is reserved allocating these slots to connections. For example, if we use a table with 100 slots for a time frame of 1μs, each slot will be allocated for 1/100 from 1μs = 10ns. If the network provides 1Gb/s per link, the bandwidth per slot will be 1/100 from 1Gbs = 10Mb/s. We can only allocate multiple of 10Mb/s for guaranteed throughput traffic.
For a read command generating long bursts, allocating the minimum bandwidth of 10Mb/s would be probably to much, as it will use only a small fraction of it. The bandwidth can indeed be used by best-effort traffic, however, not by other guaranteed throughput traffic. As a result, not all the traffic for which guarantees are needed may fit in the slot table.
An alternative is to use more slots, but this increases the cost of the router. This is why, a best effort command may be a better solution.
Page: 8

4. This definition is good for outgoing messages, as there is one source (ANIP) and potentially multiple destinations (PNIPs). However, for return messages, we define global/local ordering as follows. Global ordering means that responses from all PNIPs/slaves (i.e. sources of messages in this case) com in the same order as the transactions have been initiated (i.e., the same order as the commands have been issued by the master to the ANIP). Local ordering guarantees the order of response only if they come from the same slave/PNIP.
5Page: 8

Slave modules
Page: 8

6. We can only guarantee the order we offer transactions to the slave module, but the order of processing depends on the module implementation. It can well decide to process transactions in a different order (e.g., memory controller). For ordering we only require the responses are returned in the same order as the transactions were accepted.
Page: 8

7. This is only valid for global ordering. For local ordering (i.e., order preserved only per slave), if ordered transport channels are used, no sorting is necessary.
Page: 8

8. Global ordering of responses conforms with AMBA. Local ordering of responses does not.
Page: 8

9. I think Kees meant write transactions may be critical and we don't want to loose them, but read transactions can be lost, because they can be tried later. See example below in the text.
Page: 8

10. The two commands (i.e., read and write) can indeed be sent from the same module. If we set up a connection with flow control for the outgoing part both commands will be delivered. However, if the return part has no flow control, the responses for read commands may be lost. In such a case, the read transactions will fail. I think Kees meant read transactions being lost, not read commands being lost.
Page: 8

11. fc = flow control, nofc = no flow control
Page: 8

12. Buffer is reserved only for a return status message, such as an acknowledgment, or an error message. Buffer can be, but is not necessarily reserved also for returned data.

08.10.2002

Page: 8
13. Data can also be dropped at the ANIP (i.e., RETDATA) when no flow control is implemented for the return part. In such a case, a RETSTAT=RETLOST will replace the RETSTAT=RETOK which accompanied the dropped RETDATA.
Page: 8
14. Has reserved
Page: 8
15. Yes, this is true. Between routers, there is always link level flow control and no data is never lost. Data can be lost only in the network interfaces, if no end-to-end flow control (here referred simply as flow control) is implemented. Therefore, here, messages reach the PNIP even when no (end-to-end) flow control is implemented.

These and other aspects are described in more detail in the following three annexes
1. Communication Services for Networks on Chip, pages 1-25 by Andrei Rădulescu and Kees Goossens;
Further background information useful for implementing the invention can be found at:
2. Networks on Silicon: Blessing or Nightmare? pp 1-5, by Paul Wielage and Kees Goossens, (published), and
3. Trade-Offs in the Design of a Router with Combined Guaranteed and Best-Effort Services for Networks on Chip, pp 1-6, by Edwin Rijpkema, Kees Goossens, Andrei Rădulescu, Jef van Meerbergen, and Paul Wielage, submitted to and rejected by ISSS2002.

I. Introduction

Networks on chip (NoC) have received considerable attention recently as a solution to the interconnect problem in highly-complex chips [3–5, 7–9, 15, 19, 22]. The reason is twofold. First, NoCs help resolve the electrical problems in new deep-submicron technologies, as they structure and manage global wires [3–5, 7, 8]. At the same time they share wires, lowering their number and increasing their utilization [7, 8]. NoCs can also be energy efficient and reliable [4], and are scalable compared to buses [9]. Second, NoCs also decouple computation from communication, which is essential in managing the design of billion-transistor chips [14, 22]. NoCs achieve this decoupling because they are traditionally designed using protocol stacks [21], which provide well-defined interfaces separating communication service usage from service implementation [5, 22].

Using networks for on-chip communication when designing systems on chip (SoC), however, raises a number of new issues that must be taken into account. This is because, in contrast to existing on-chip interconnects (e.g., buses, switches, or point-to-point wires), where the communicating modules are directly connected, in a NoC the modules communicate remotely via network nodes. As a result, interconnect arbitration changes from centralized to distributed, and issues like out-of order transactions, higher latencies, and end-to-end flow control must be handled either by the intellectual property block (IP) or by the network itself.

Most of these topics have been already the subject of research in the field of computer networks [24] and parallel machine interconnect networks [6]. However, on-chip networks have different properties (e.g., tighter link synchronization) and constraints (e.g., higher memory cost) leading to different design choices, which in the end affect the network services.

In this paper, we compare NoCs and off-chip networks showing both their similarities and differences. We also explore the differences between NoCs and existing on-chip interconnects. We list new issues that must be resolved in system design due to the multi-hop nature of NoCs, and present an interface which takes these issues into consideration. Our interface are aimed at being similar to a split-transaction bus interface, such as VCI [25] or OCP [17], to allow simple, low-cost wrappers to bus interfaces, and to allow backward compatibility with existing IPs. Our interface uses a

request-response protocol that provides basic read and write operations. But our interface extends bus interfaces to fully exploit the power of our NoC [8, 19, 20]. For example, it offers connection-based communication where end-to-end flow control and time-related guarantees (e.g., bounded latency) can be requested.

The paper is organized as follows. In the next two sections we compare NoCs properties with those of off-chip networks and buses, respectively. In Section IV, we present the services we offer in our network. Finally, we present our conclusions.

II. Networks Brought on Chip

Networks have been the subject of research for decades, both in the context of local and wide area networks (computer networks) [24], and as an interconnect for parallel machines [6]. Both are very much related to on-chip networks, and many of the results in those fields are also applicable on chip. However, NoC's premises are different from off-chip networks, and, therefore, most of the network design choices must be reevaluated.

NoCs differ from off-chip networks mainly in their constraints and synchronization. Typically, most on-chip resources have much tighter constraints compared to off-chip. Storage (i.e., memory) and computation resources are relatively more expensive, whereas the number of point-to-point links is larger on chip than off chip [7].

Storage is expensive, because general-purpose on-chip memory, such as RAMs, occupy a large area. Having the memory distributed in the network components in relatively small sizes is even worse, as the of overhead area in the memory then becomes dominant.

Also computation for on-chip networks comes at a relatively high cost compared to off-chip networks. An off-chip network interface usually contains a dedicated processor to implement the protocol stack up to network layer or even higher, to off-load the host processor from the communication processing. Including a dedicated processor in a network interface is not feasible on chip, as the size of the network interface will become comparable to or larger than the IP to be connected to the network. Moreover, running the protocol stack on the IP itself may also be not feasible, be-

cause often these IPs have one dedicated function only, and do not have the capabilities to run a network protocol stack.

The number of wires and pins to connect network components is an order of magnitude larger on chip than off chip [7]. If they are not used massively for other purposes than NoC, they allow wide point-to-point interconnects (e.g., 300-bit links) [7, 15]. This is not possible off-chip, where links are relatively narrower: 8-16 bits.

On-chip wires are also relatively short, allowing a much tighter synchronization than off chip. This allows a reduction in the buffer space in the routers because the communication can be done at a smaller granularity. In the current semiconductor technologies, wires are also fast and reliable, which allows simpler link-layer protocols (e.g., no need for error correction, or retransmission). This also compensates for the lack of memory and computational resources.

In the rest of the section, we list five network issues that have a direct impact on the NoC cost: reliable communication, deadlock, data ordering, network flow control and buffering strategy, and time-related guarantees. For each of them, we discuss the differences and similarities for on- and off-chip networks.

Reliable communication. A consequence of the tight on-chip resource constraints is that the network components (i.e., routers and network interfaces) must be fairly simple to minimize computation and memory requirements. Luckily, on-chip wires provide a reliable communication medium, which avoids the considerable overhead incurred by the off-chip networks for providing reliable communication. Data integrity can be provided at low cost at the data link layer. However, data loss also depends on the network architecture, as in most computer networks data is simply dropped if congestion occurs in the network [6,24]. On-chip, dropping data may lead to a too costly implementation of reliable communication. We show below that a network where no data is dropped can lead to a much lower-cost solution, at the peril of introducing the possibility of deadlock.

Deadlock. Computer network topologies have generally an irregular (possibly dynamic) structure and bidirectional links, which can introduce buffer cycles. In such topologies, packet dropping at the network nodes

may be required to avoid deadlocks.

Deadlock can also be avoided without dropping data, for example, by introducing constraints either in the topology or routing. Fat-tree topologies have already been considered for NoCs, where deadlock is avoided by bouncing back packets in the network in case of overflow [9]. Tile-based approaches to system design [7, 15, 23] use mesh or torus network topologies, where deadlock can be avoided using, for example, a turn-model routing algorithm [6].

An alternative solution for deadlock in NoCs, which takes into consideration that modules connecting to the network are either masters (initiating requests and receiving responses), or slaves (receiving requests and sending back responses), is to maintain separate virtual networks (with separate buffers) for requests and responses [6].

Data ordering. In a network, data sent from a source to a destination may arrive out of order due to reordering in network nodes, following different routes, or retransmission after dropping. For off-chip networks our-of-order data delivery is typical. However, for NoCs where no data is dropped, data can be forced to follow the same path between a source and a destination (deterministic routing) with no reordering. This in-order data transportation requires less buffer space, and reordering modules are no longer necessary.

Network flow control and buffering strategy. Network flow control and buffering strategy have a direct impact on the memory utilization in the network. Wormhole routing requires only a flit buffer in the router, whereas store-and-forward and virtual-cut-through routing require at least the buffer space to accommodate a packet. Consequently, on chip, wormhole routing may be preferred over virtual-cut-through or store-and-forward routing. Similarly, input queuing may be a lower memory-cost alternative to virtual-output-queuing or output-queuing buffering strategies, because it has fewer queues. Dedicated fifo memory structures at a low cost also enable on-chip usage of virtual-cut-through routing or virtual output queuing for a better performance [19]. However, using virtual-cut-through routing and virtual output queuing at the same time is still too costly [19].

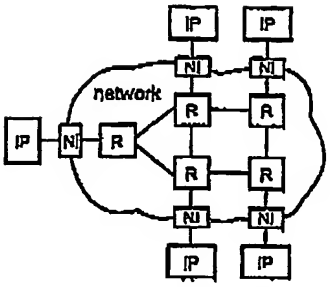

Figure 1. A network interconnect example

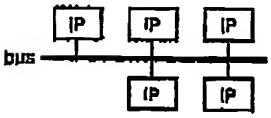

Figure 2. A bus interconnect example

Time-related guarantees. Off-chip networks typically use packet switching and offer best-effort services. Contention can occur at each network node, making latency guarantees very hard to offer. Throughput guarantees can still be offered using schemes such as rate-based switching [26] or deadline-based packet switching [18], but with high buffering costs.

An alternative to provide such time-related guarantees is to use time-division multiple access (TDMA) circuits, where every circuit is dedicated to a network connection. Circuits provide guarantees at a relatively low memory and computation cost. Network resource utilization is increased when the network architecture allows any left-over guaranteed bandwidth to be used by best-effort communication [10, 19, 20].

III. From buses to NoCs

Introducing networks (Figure 1) as on-chip interconnects radically changes the communication when compared to direct interconnects, such as buses or switches (Figure 2). This is because of the multi-hop nature of a network, where communication modules are not directly connected, but separated by one or more network nodes. This is in contrast with the prevalent existing interconnects (i.e., buses) where modules are directly connected. The implications of this change reside in the arbitration (which must change from centralized to distributed), and in the communication properties (e.g., ordering, or flow control).

In this section, we list some of these topics, and outline the differences of NoCs and buses. We refer mainly to buses as direct interconnects, because currently they are the most used on-chip interconnect. Most of the bus characteristics also hold for other direct interconnects (e.g., switches [16]). Multilevel buses are a hybrid between buses and NoCs. Depending on the functionality of the bridges, for our purposes, multilevel buses either behave like simple buses [2] or like NoCs.

Programming Model. The programming model of a bus typically consists of load and store operations which are implemented as a sequence of primitive bus transactions. Bus interfaces typically have dedicated groups of wires for command, address, write data, and read data [1, 12, 13, 17, 25].

A bus is a resource shared by multiple IPs. Therefore, before using it, IPs must go through an arbitration phase, where they request access to the bus, and block until the bus is granted to them.

A bus transaction involves a request and possibly a response. Modules issuing requests are called masters, and those serving requests are called slaves. If there is a single arbitration for a pair of request-response, the bus is called non-split. In this case, the bus remains allocated to the master of the transaction until the response is delivered, even when this takes a long time. Alternatively, in a split bus, the bus is released after the request to allow transactions from different masters to be initiated. However, a new arbitration must be performed for the response such that the slave can access the bus [11].

For both split and non-split buses, both communication parties have direct and immediate access to the status of the transaction. In contrast, network transactions are one-way transfers from an output buffer at the source to an input buffer at the destination that causes some action at the destination, the occurrence of which is not visible at the source [6]. The effects of a network transaction are observable only through additional transactions. A request-response type of operation is still possible, but requires at least two distinct network transactions. Thus, a bus-like transaction in a NoC will essentially be a split transaction.

Transaction Ordering. Traditionally, on a bus all transactions are ordered (cf. Peripheral VCI [25], AMBA [1], or CoreConnect PLB and OPB [12, 13]). This is possible at a low cost, because the interconnect, being a direct link between the communicating parties, does not reorder of data. However, on a split bus, a total ordering of transactions on a single master may still cause performance penalties, when slaves respond at different speeds. To solve this problem, recent extensions to bus protocols allow transactions to be performed on connections. Ordering of transactions within a connection is still preserved, but between connections there are no ordering constraints (e.g., OCP [17], or Basic VCI [25]). A few of the bus protocols allow out-of-order responses per connection in their advanced modes (e.g., Advanced VCI [25]), but both requests and responses arrive at the destination in the same order as they were sent.

In a NoC, ordering becomes weaker. Global ordering can only be provided at a very high cost due to the conflict between the distributed nature of the networks, and the requirement of a centralized arbitration necessary for global ordering.

Even local ordering, between a source-destination pair, may be costly. Data may arrive out of order if it is transported over multiple routes. In such cases, to still achieve an in-order delivery, data must be labeled with sequence numbers and reordered at the destination before being delivered.

Atomic Chains of Transactions. An atomic chain of transactions is a sequence of transactions initiated by a single master that is executed on a single slave exclusively. That is, other masters are denied access to that slave, once the first transaction in the chain claimed it. This mechanism is widely used to implements synchronization mechanisms between master modules (e.g., semaphores).

On a bus, atomic operations can easily be implemented, as the central arbiter will either (a) lock the bus for exclusive use by the master requesting the atomic chain, or (b) know not to grant access to a locked slave. In the former case, the time resources are locked is shorter because once a master has been granted access to a bus, it can quickly perform all the transactions in the chain (no arbitration delay is required for the subsequent transactions in the chain). Consequently, the locked slave and the bus can be opened up again in a short time. This approach is used in AMBA, and

CoreConnect. In the latter case, the bus is not locked, and can still be used by other modules, however, at the price of a longer locking time of the slave. This approached is used in VCI and OCP.

In a NoC, where the arbitration is distributed, masters do not know that a slave is locked. Therefore, transactions to a locked slaved may still be initiated, even though the locked slave cannot accept them. Consequently, to prevent deadlock, these other transactions must be either dropped, or stored such that transactions in the atomic chain can be filtered and still be served. Moreover, the time a module is locked is much longer in case of NoCs, because of the higher latency per transaction.

Deadlock. In buses, the deadlocks are not generally an issue. Deadlock can still occur at the application level (e.g., an atomic chain of transactions that locks the bus, which is never finished), but it is not caused by the interconnect itself.

In a network, deadlock becomes a more important issue, and special care has to be taken in the network design to avoid deadlock. Deadlock is mainly caused by cycles in the buffers. To avoid deadlock, either network nodes must drop packets when their buffer are filled, or routing must be cycle-free. In a NoC, we believe the latter is preferable, because of its lower cost in achieving reliable communication (see Section II).

A second cause of deadlock are atomic chains of transactions. The reason is that while a module is locked, the queues storing transactions may get filled with transactions outside the atomic transaction chain, blocking the access of the transaction in the chain to reach the locked module. If atomic transaction chains must be implemented (to be compatible with processors allowing this, such as MIPS), the network nodes should be able to filter the transactions in the atomic chain, or be allowed to drop those blocking them.

Media Arbitration. An important difference between buses and NoCs is in the media arbitration scheme. In a bus, master modules request access to the interconnect, and the arbiter grants the access for the whole interconnect at once. Arbitration is *centralized* as there is only one arbiter component, and *global* as all the requests as well as the state of the interconnect are visible to the arbiter. Moreover, when a grant is given, the

complete path from the source to the destination is exclusively reserved.

In a non-split bus, arbitration takes place once when a transaction is initiated. As a result, the bus is granted for both request and response. In a split bus, requests and responses are arbitrated separately.

In a NoC arbitration is also necessary, as it is a shared interconnect. However, in contrast to buses, the arbitration is *distributed*, because it is performed in every router, and is based only on *local* information. Arbitration of the communication resources (links, buffers) is performed incrementally as the request or response advances [19].

Destination Name and Routing. For a bus, the command, address, and data are broadcasted on the interconnect. They arrive at every destination, of which one activates based on the broadcasted address, and executes the requested command. This is possible because all modules are directly connected to the same bus.

In a NoC, it is not feasible to broadcast information to all destinations, because it must be copied to all routers and network interfaces. This floods the network with data. The address is better decoded at the source to find a route to the destination module. A transaction address will therefore have two parts: (a) a destination identifier, and (b) an internal address at the destination.

Latency. Transaction latency is caused by two factors: (a) the access time to the bus, which is the time until the bus is granted, and (b) the latency introduced by the interconnect to transfer the data.

For a bus, where the arbitration is centralized the access time is proportional to the number of masters connected to the bus. The transfer latency itself typically is constant and relatively fast, because the modules are linked directly. However, the speed of transfer is limited by the bus speed, which is relatively slow for buses.

In a NoC, arbitration is performed at each router for the following link. The access time per router is small. Both end-to-end access time and transport time increase proportionally to the number of hops between master and slave. However, network links are unidirectional and point to point, and hence can run at higher frequencies than buses, thus lowering the latency.

From a latency prospective, using a bus or a network is a trade off between the number of modules connected to the interconnect (which affects access time), the speed of the interconnect, and the network diameter.

Data Format. In most modern bus interfaces the data format is defined by separate wire groups for the transaction type, address, write data, read data, and return acknowledgments/errors (e.g., VCI, OCP, AMBA, or CoreConnect). This is used to pipeline transactions. For example, concurrently with sending the address of a read transaction, the data of a previous write transaction can be sent, and the data from an even earlier read transaction can be received. Moreover, having dedicated wire groups simplifies the transaction decoding; there is no need for a mechanism to select between different kinds of data sent over a common set of wires.

Inside a network, there is typically no distinction between different kinds of data. Data is treated uniformly, and passed from one router to another. This is done to minimize the control overhead and buffering in routers. If separate wires would be used for each of the above-mentioned groups, separate routing, scheduling, and queuing would be needed, increasing the cost of routers.

In addition, in a network at each layer in the protocol stack, control information must be supplied together with the data (e.g., command type, address, or data size). This control information is organized as an envelope around the data. That is, first a header is sent, followed by the actual data (payload), followed possibly by a trailer. Multiple such envelopes may be provided for the same data, each carrying the corresponding control information for each layer in the network protocol stack [6,24].

Buffering and Flow Control. Buffering data of a master (output buffering) is used both for buses and NoCs to decouple computation from communication. However, for NoCs output buffering is also needed to marshal data, which consists of (a) (optionally) splitting the outgoing data in smaller packets which are transported by the network, and (b) adding control information for the network around the data (packet header). To avoid output buffer overflow the master must not initiate transactions that generate more data than the currently available space.

Similarly to output buffering, input buffering is also used to decouple

computation from communication. In a NoC, input buffering is also required to unmarshal data.

In addition, flow control for input buffers differs for buses and NoCs. For buses, the source and destination are directly linked, and, destination can therefore signal directly to a source that it cannot accept data. This information can even be available to the arbiter, such that the bus is not granted to a transaction trying to write to a full buffer.

In a NoC, however, the destination of a transaction cannot signal directly to a source that its input buffer is full. Consequently, transactions to a destination can be started, possibly from multiple sources, after the destination's input buffer has filled up. Two policies can be adopted when an input buffer is full. The first is not to accept additional incoming transitions, and to store them in the network. However, this approach can easily lead to network congestion, as the data could be eventually stored all the way to the sources, blocking the links in between. The second approach is to accept incoming transactions at a full destination, and drop some data in the input buffer. Congestion is avoided but data is lost, which is undesirable.

To avoid output buffer overflow connections can be used, together with end-to-end flow control. At connection set up between a master and one or more slaves, buffer space is allocated at the network interfaces of the slaves, and the network interface of the master is assigned credits reflecting the amount of buffer space at the slaves. The master can only send data when it has enough credits for the destination slave(s). The slaves grant credits to the master when they consume data.

IV. The Æthereal Approach

As described in the previous two sections, NoCs have different properties from both existing off-chip networks and existing on-chip interconnects. As a result, existing protocols and service interfaces cannot be adopted directly to NoCs, but must take the characteristics of NoCs into account. For example, a protocol such as TCP/IP assumes the network is lossy, and includes significant complexity to provide reliable communication. Therefore, it is not suitable in a NoC where we assume data transfer

reliability is already solved at a lower level. On the other hand, existing on-chip protocols such as VCI, OCP, AMBA, or CoreConnect are also not directly applicable. For example, they assume ordered transport of data: if two requests are initiated from the same master, they will arrive in the same order at the destination. This does not hold automatically for NoCs. Atomic chains of transactions and end-to-end flow control also need special attention in a NoC interface.

Our objectives when defining our network services are the following. First, the services abstract from the network internals as much as possible. This is a key ingredient in tackling the challenge of *decoupling the computation from communication* [14,22], which allows IPs (the computation part), and the interconnect (the communication part) to be designed independently from each other. As a consequence, our services are positioned at the transport layer in the ISO-OSI reference model [24], which is the first layer to be independent of the implementation of the network.

Second, we aim at a NoC interface as close as possible to a bus interface. NoCs can then be introduced non-disruptively: with minor changes, existing IPs, methodologies and tools can continue to be used. As a consequence, we use a request-response interface, similar to interfaces for split buses [1, 12, 13, 17, 25].

Third, our interface extends traditional bus interfaces to fully exploit the power of NoCs. For example, we offer connection-based communication which does not only relax ordering constraints (as for buses), but also enables new communication properties, such as end-to-end flow control based on credits, or guaranteed throughput [8, 19, 20]. All these properties can be set for each connection individually.

A. The Æthereal Connection and Transaction Model

IPs interact with our network [8, 19, 20] at so-called network interfaces (NI). NIs provide NI ports (NIP) through which the communication services are accessed. As shown in Figure 3, a NI can have several NIPs to which one or more IPs (computation elements or memories, but not interconnection elements) can be connected. Similarly, an IP can be connected to more than one NIs and NIPs.

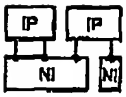

Figure 3. Examples of links between NIs and IPs.

Communication between NIPs is performed on *connections*. Connections are introduced to describe and identify communication with different properties, such as guaranteed throughput, bounded latency and jitter, ordered delivery, or flow control. For example, to distinguish and independently guarantee communication of 1Mbs and 25Mbs, two connections can be used. Two NIPs can be connected by multiple connections, possibly with different properties. Connections as defined here are similar to the concept of threads and connections from OCP and VCI. Where in OCP and VCI connections are used only to relax transaction ordering, we generalize from only the ordering property to include configuration of buffering and flow control, guaranteed throughput, and bounded latency per connection.

Æthereal connections must be *created* with the desired properties before being used. This may result in *resource reservations* inside the network (e.g., buffer space, or percentage of the link usage per time unit). If the requested resources are not available, the network will refuse the request. After usage, connections are *closed*, which leads to freeing the resources occupied by that connection.

To allow more flexibility in configuring connections, and, hence, better resource allocation per connection, the outgoing and return parts of connections are configured separately. For example, different buffer space can be allocated in the ANIP and PNIPs, respectively, or different bandwidths can be reserved for requests and responses.

Depending on the requested services, the time to handle a connection (i.e., creating, closing, modifying services) can be short (e.g., creating/closing an unordered, lossy, best-effort connection) or significant (e.g., creating/closing a multicast guaranteed-throughput connection). Consequently, connections are assumed to be created, closed, or modified infrequently, coinciding e.g. with reconfiguration points, when the application requirements change.

Communication takes place on connections using *transaction*, consisting of a request and a possibly response. The request encodes an operation

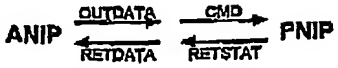

Figure 4. Transaction composition.

(e.g., read, write, flush, test and set, nop) and possibly carries outgoing data (e.g., for write commands). The response returns data as a result of a command (e.g., read) and/or an acknowledgment.

Connections involves at least two NIPs. Transactions on a connection are always started at one and only one of the NIPs, called the connection's *active* NIP (ANIP). All the other NIPs of the connection are called *passive* NIPs (PNIP).

There can be multiple transactions active on a connection at a time (as for split buses). That is, transactions can be started at the ANIP of a connection while responses for earlier transactions are pending. If a connection has multiple slaves, multiple transactions can be initiated towards different slaves. Transactions are also pipelined between a single pair of a master and a slave for both requests and responses. In principle, transactions can also be pipelined within a slave, if the slave allows this.

A transaction is composed from the following messages (see Figure 4):

- A *command* message (CMD) is sent by the ANIP, and describes the action to be executed at the slave connected to the PNIP. Examples of commands are read, write, test and set, and flush. Commands are the only messages that are compulsory in a transaction. For NIPs that allow only a single command with no parameters (e.g., fixed-size address-less write), we assume the command message still exists, even if it is implicit (i.e., not explicitly sent by the IP).
- An *out data* message (OUTDATA) is sent by the ANIP following a command that requires data to be executed (e.g., write, multicast, and test-and-set).
- A *return data* message (RETDATA) is sent by a PNIP as a consequence of a transaction execution that produces data (e.g., read, and test-and-set).
- A *completion acknowledgment* message (RETSTAT) is an optional message which is returned by PNIP when a command has been completed. It may signal either a successful completion or an er-

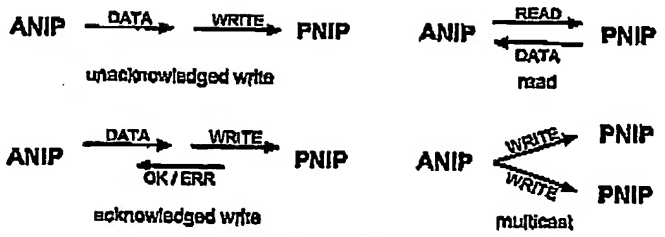

Figure 5. Transaction examples.

ror. For transactions including both RETDATA and RETSTAT the two messages can be combined in a single message for efficiency. However, conceptually, they exist both: RETSTAT to signal the presence of data or an error, and RETDATA to carry the data. In bus-based interfaces RETDATA and RETSTAT typically exist as two separate signals [1, 12, 13, 17, 25].

Messages composing a transaction are divided in *outgoing* messages, namely CMD and OUTDATA, and *response* messages, namely RETDATA, RETSTAT. Within a transaction, CMD precedes all other messages, and RETDATA precedes RETSTAT if present. These rules apply both between master and ANIP, and PNIP and slave. Examples of transactions are shown in Figure 5.

We classify connections as follows (see Figure 6):

- A *simple* connection is a connection between one ANIP and one PNIP.
- A *narrowcast* connection is a connection between one ANIP and one or more PNIPs, in which the ANIP initiates transactions that are executed by exactly one PNIP. An example of the narrowcast connection is shown in Figure 7, where the ANIP performs transactions on an address space which is mapped on two memory modules. Depending on the transaction address, a transaction is executed on only one of these two memories.
- A *multicast* connection is a connection between one ANIP and one or more PNIPs, in which the sent messages are duplicated and each PNIP receives a copy of those messages. In a multicast con-

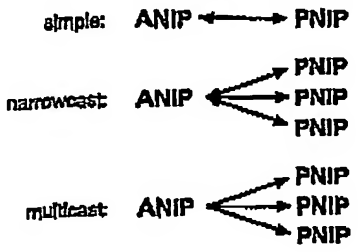

Figure 6. Connection types.

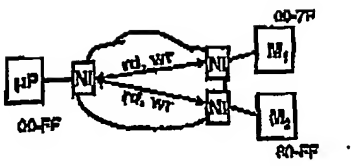

Figure 7. A narrowcast connection.

nection no return messages are currently allowed, because of the large traffic they generate (i.e., one response per destination). It could also increase the complexity in the ANIP because individual responses from PNIPs must be merged into a single response for the ANIP. This requires buffer space and/or additional computation for the merging itself.

B. Connection Properties

In this section we describe the properties that can be configured for a connection: guaranteed message integrity, guaranteed transaction completion, various transaction orderings, guaranteed throughput, bounded latency and jitter, and connection flow control.

Data Integrity. Data integrity means that the payload of the message is not changed (accidentally or not) during transport. We assume that data integrity is already solved at a lower layer in our network, namely at the link layer, because in current on-chip technologies data can be transported uncorrupted over links. Consequently, our network interface always guarantees that messages are delivered uncorrupted at the destination.

Transaction Completion. A transaction without a response is said to be complete when it has been executed by the slave. As there is no response message to the master, no guarantee regarding transaction completion can

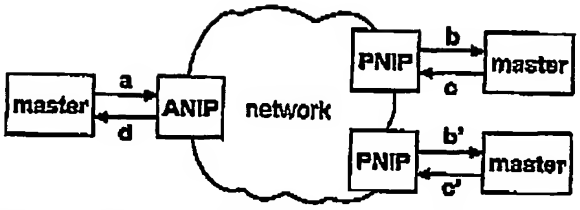

Figure 8. Message ordering is observable at a, b, c, and d.

be given.

A transaction with a response is said to be complete when a RETSTAT message is received from the ANIP[1]. The transaction may either (a) be executed successfully, in which case a success RETSTAT is returned, (b) fail in its execution at the slave, and then an execution error RETSTAT is returned, or (c) fail because of buffer overflow in a connection with no flow control, and then it reports an overflow error.

In our network, routers do not drop data [20], therefore, messages are always guaranteed to be delivered at the NI. For connections with flow control, also NIs do not drop data. Thus, message delivery to the IPs is guaranteed automatically in this case.

However, if there is no flow control, messages may be dropped at the network interface in case of buffer overflow (see the paragraph on end-to-end flow control below). All of CMD, OUTDATA, and RETDATA may be dropped at the NI. To guarantee transaction completion, RETSTAT is not allowed to be dropped. Consequently, in the ANIPs enough buffer space must be provided to accommodate RETSTAT messages for all outstanding transactions. This is enforced by bounding the number of outstanding transactions.

Transaction Ordering. In this section, we describe the ordering requirements between different transactions within a single connection. Over different connections no ordering of transactions is defined at the transport layer.

[1] We assume that when data is received as a response (RETDATA), a RETSTAT (possibly implicit) is also received to validate the data.

There several points in a connection where order of transactions can be observed (see Figure 8): (a) the order in which the master presents CMD messages to the ANIP, (b) the order in which the CMDs are delivered to the slave by the PNIP, (c) the order in which the slave presents the responses to the PNIP, and (d) the order the responses are delivered to the master by the ANIP. Note that not all of (b), (c), and (d) are always present. Moreover, there are no assumptions about the order in which the slaves execute transactions; we can only observe the order of the responses. We consider the order of the transaction execution to be a system decision, and not a part of the interconnect protocol.

At both ANIP and PNIPs, outgoing messages belonging to different transactions on the same connection are allowed to be interleaved. For example, two write commands can be issued, and only afterwards their data follows. If the order of OUTDATA messages differs from the order of CMD messages, transaction identifiers must be introduced to associate OUTDATAs with their corresponding CMD.

Outgoing messages can be delivered by the PNIPs to the slaves (see Figure 8-b) as follows:

- *Unordered*, which imposes no order on the delivery of the outgoing messages of different transactions at the PNIPs.
- *Ordered locally*, where transactions must be delivered to each PNIP in the order they were sent, but no order is imposed across PNIPs. Locally-ordered delivery of the outgoing messages can be provided either by an ordered data transportation, or by reordering of outgoing messages at the PNIP.
- *Ordered globally*, where transactions must be delivered in the order they were sent, across all PNIPs of the connection. Globally-ordered delivery of the outgoing part of transactions require a costly synchronization mechanism.

Transaction response messages can be delivered by the slaves to the PNIPs (see Figure 8-c) as follows:

- *Ordered*, when RETDATA and RETSTAT messages are returned in the same order as the CMDs were delivered to the slave.
- *Unordered*, otherwise.

When responses are unordered, there has to be a mechanism to identify the transaction to which a response belongs. This is usually done using tags attached to messages for transaction identifications (similar to tags in VCI).

Response messages can be delivered by the ANIP to the master (see Figure 8-d) as follows:

- *Unordered*, which imposes no order on the delivery of responses. Here, also, tags must be used to associate responses with their corresponding CMDs.
- *Ordered locally*, where RETDATA and RETSTAT messages of transactions for a single slave are delivered in the order the original CMDs were presented by the master to the ANIP. Note that there is no ordering imposed for transactions to different slaves within the same connection.
- *Globally ordered*, where all responses in a connection are delivered to the master in the same order as the original CMDs. When transactions are pipelined on a connection, then globally-ordered delivery of responses requires reordering at the ANIP.

All $3 \times 2 \times 3 = 18$ combinations between the above orderings are possible. Out of these, we define and offer the following two. An *unordered* connection is a connection in which no ordering is assumed in any part of the transactions. As a result, the responses must be tagged to be able identify to which transaction they belong. Implementing unordered connections has low cost, however, they may be harder to use, and introduce the overhead of tagging.

An *ordered* connection is defined as a connection with *local* ordering for the outgoing messages from PNIPs to slaves (Figure 8-b), *ordered* responses at the PNIPs (Figure 8-c), and *global* ordering for responses at the ANIP (Figure 8-d). We choose local ordering for the outgoing part because the global ordering has a too high cost, and has few uses. The ordering of responses is selected to allow a simple programming model with no tagging. Global ordering at the ANIP is possible at a moderate cost, because all the ordering is done locally in the ANIP.

A user can emulate connections with global ordering at the PNIPs using non-pipelined acknowledged transactions.

Connection latency, throughput, and jitter. In our network, throughput can be reserved for connections in a time-division multiple access (TDMA) fashion, where bandwidth is split in fixed-size slots on a fixed time frame. Bandwidth, as well as bounds on latency and jitter can be guaranteed when slots are reserved. They are all defined in multiples of the slots.

Guaranteed-throughput connections can overbook resources in some cases. For example, when an ANIP opens a guaranteed-throughput read connection, it must reserve slots for the read command messages, and for the read data messages. The ratio between the two can be very large (e.g., 1:100), which leads either to a large number of slots, or bandwidth being wasted for the read command messages.

To solve this problem, we allow the request and response parts of a connection be configured independently for all of throughput, latency and jitter. Consequently, the request part of a connection can be best effort, while the response can have guaranteed throughput (or vice versa). For the example mentioned above, we can use best effort read messages, and guaranteed-throughput read-data messages. No global connection guarantees can be offered in this case, but the overall throughput can be higher and more stable than in the case of using only best-effort traffic.

Connection flow control. As mentioned earlier, our network guarantees that messages are delivered to the NI. Messages sent from one of the NIPs are not immediately visible at the other NIP, because of the multi-hop nature of networks. Consequently, handshakes over a network would allow only a single message be transmitted at a time. This limits the throughput on a connection and adds latency to transactions. To solve this problem, and achieve a better network utilization, the messages must be pipelined. In this case, if the data is not consumed at the PNIP at the same rate it arrives, either flow control must be introduced to slow down the producer, or data may be lost because of limited buffer space at the consumer NI.

We introduce end-to-end flow control at the level of connections, which requires buffer space to be associated with connections. End-to-end flow control ensures that messages are sent over the network only when there is enough space in the NIP's destination buffer to accommodate them.

End-to-end flow is optional (i.e., to be requested when the connections

opened) and can be configured independently for the outgoing and return paths. When no flow control is provided, messages are dropped when buffers overflow. Multiple policies of dropping messages are possible, as in off-chip networks. Possible scenarios include: (a) the oldest message is dropped (milk policy), or (b) the newest message is dropped (wine policy) [24].

We opt for a credit-based flow control. Credits are associated with the empty buffer space at the receiver NI. The sender's credit is lowered as data is sent. When data is delivered at the receiver NIP, credits are granted to the sender. If the sender's credit is not sufficient to send some data, the NI at the sender stalls the sending.

C. Use Cases

To illustrate the need for differentiated services on connections, we show in this section some examples of traffic. We describe the properties they would use over an Æthereal connection to meet their traffic requirements.

Video processing streams typically require a lossless, in-order video stream with guaranteed throughput, but possibly allow corrupted samples. An Æthereal connection for such a stream would require the necessary throughput, ordered transactions, and flow control. If the video stream is produced by the master, only write transactions are necessary. In such a case, with a flow-controlled connection there is no need to also require transaction completion, because messages are never dropped, and the write command and its data are always delivered at the destination. Data integrity is always provided by our network, even though it may be not necessary in this case.

Another example is that of cache updates which require uncorrupted, lossless, low-latency data transfer, but ordering and guaranteed throughput are less important. In such a case, a connection would not require any time related guarantees, because a low latency, even if preferable, is not critical. Low latency can be obtained even with a best effort connection. The connection would also require flow control and guaranteed transaction completion to ensure loss-less transactions. However, no ordering is

necessary, because this is not important for cache updates, and allowing out of order transaction can reduce the response time.

V. Conclusions

In this paper, we compare networks on chip (NoC) to off-chip networks (e.g., computer networks) and existing on-chip interconnects (e.g., busses). We show that NoCs have many similarities with off-chip networks. However, they also differ, especially in their resource constraints. For example on a chip, memory and computation resources are more expensive, while there are more wires. This makes NoC architectures different from off-chip networks, and requires rethinking of network services.

We also compare NoCs to existing on-chip interconnects, such as buses and switches. By directly connecting IP blocks, existing on-chip interconnects can offer tight coupling between masters and slaves, and global arbitration. In NoCs, masters and slaves are completely decoupled, and the arbitration is distributed over the network nodes. This make it harder to provide guarantees, such as bandwidth lower bounds, and transaction orderings.

We define a set of NoC services that abstract from the network details. Using these services in the IP design decouples computation and communication. We use a request-response transaction model to be close to existing on-chip interconnect protocols. This eases the migration of current IPs to NoCs. To fully utilize the NoC capabilities, such as high bandwidth and transaction concurrency, our services provide connection-oriented communication. Connections can be configured independently with different properties. These properties include transaction completion, various transaction ordering, bandwidth lower bounds, latency and jitter upper bounds, and flow control.

Our services are a prerequisite for service-based system design which makes applications independent of NoC implementations, makes designs more robust, and enables architecture-independent quality-of-service strategies.

References

1. ARM. AMBA specification. rev. 2.0, 1999.
2. ARM. Multi-layer AHB. overview., 2001.
3. J. Bainbridge and S. Furber. CHAIN: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5), 2002.
4. L. Benini and G. De Micheli. Powering networks on chips. In *ISSS*, 2001.
5. L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.
6. D. J. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.
7. W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC*, 2001.
8. K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *DATE*, 2002.
9. P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE*, 2000.
10. P. J. Having. *Mobile Multimedia Systems*. PhD thesis, University of Twente, 2000.
11. J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1995.
12. IBM. 32-bit on-chip peripheral bus. rev. 2.1, 2001.
13. IBM. 32-bit processor local bus. rev. 2.9, 2001.
14. K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.
15. S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, Tiensyrjä, and A. Hemani. A network on chip architecture and design methodology. In *ISVLSI*, 2002.
16. J. A. Leijten, J. L. van Meerbergen, A. H. Timmer, and J. A. Jess. Prophid, a data-driven multi-processor architecture for high-performance DSP. In *ED&TC*, 1997.

17. OCP International Partnership. Open core protocol specification, 2001.
18. J. Rexford. *Tailoring Router Architectures to Performance Requirements in Cut-Through Networks*. PhD thesis, Univ. Michigan, 1999.
19. E. Rijpkema, K. Goossens, A. Rădulescu, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. submitted, 2002.
20. E. Rijpkema, K. Goossens, and P. Wielage. A router architecture for networks on silicon. In *Progress*, Oct. 2001.
21. M. T. Rose. *The Open Book: A Practical Perspective on OSI*. Prentice Hall, 1990.
22. M. Sgori, M. Sheets, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC*, 2001.
23. P. Stravers and J. Hoogerbrugge. Homogeneous multiprocessing and the future of silicon design paradigms. In *VLSI-TSA*, 2001.
24. A. S. Tanenbaum. *Computer Networks*. Prentice Hall, third edition, 1996.
25. VSI Aliance. Virtual component interface standard, 2000.
26. H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. of the IEEE*, 83(10):1374–1396, 1995.

08.10.2002

Networks on Silicon: Blessing or Nightmare?

Paul Wielage and Kees Goossens
Philips Research Laboratories, Eindhoven, The Netherlands
{paul.wielage,kees.goossens}@philips.com

Abstract

Continuing VLSI technology scaling raises several deep submicron (DSM) problems like relatively slow interconnect, power dissipation and distribution, and signal integrity. Those problems are encountered particularly on long wires for global interconnect. As clock frequencies increase, scaled wires become relatively slower, and on-chip communication will be the limiting performance factor of future chips. We explain why efficiently sharing of the wires for long distance communication is the solution to this problem. We introduce networks on silicon (NoS), that route packets over shared (semi)-global wires. NoS performance is expected to be high, but comes at a cost. Balancing the performance and cost of a NoS is a major challenge, and we believe busses still have a role play.

1 Technology trend

VLSI technology scaling has long followed Moore's law. No fundamental barriers have been identified that invalidate this law for at least another decade [12]. Moore's law predicts that chips in 2010 will count over 4 billion transistors, operating in the multi-GHz range. This abundance of transistors will make very complex *systems on silicon* (SoS) possible.

However, challenges at all abstraction levels of design will have to be addressed before such SoSs will become a reality. The three most important deep submicron (DSM) challenges, related to all abstraction levels, are: substantial wire delay, controlling power delivery and dissipation, and assuring signal integrity.

Until recently, on-chip wiring was cheap. Consequently architectural models have been employed that relied on low-latency communication to globally share expensive computational resources. Global wire delay stays at best constant under technology scaling and hence these wires become effectively slower compared to a gate delay. For example, for 130 nm technology the reachable distance of a repeated global signal in a clock cycle is no more than the length of a chip [4]. For 50 nm technology, crossing a chip with highly optimized interconnect takes between six and ten clock-cycles, clearly invalidating the low-latency assumption of today. Hence we must move to system-level architectures that scale with technology.

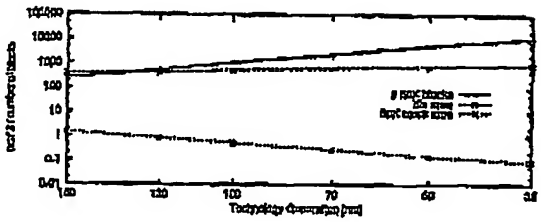

Figure 1. The number of 50k blocks for future process technologies.

A feasible template for a future-proof architecture is constructed from processing nodes that do not grow in complexity with technology. Instead, as technology scales, the number of these processing nodes on the chip grows. An on-chip communication network then combines these nodes into a SoS [4].

Various publications show that the spanning wires in blocks of 50k gates scale with technology [4, 13]. This means that the aforementioned DSM issues can be handled by CAD tools, assuming their evolutionary improvement. Figure 1 shows the exponentially increasing amount of such 50k blocks for a large die in subsequent technologies; in 35 nm this number is approximately ten thousand (adapted from [13] and [4]). It remains to find a communication architecture that allows a SoS composed of these blocks cooperate efficiently.

2 Networks on silicon are inevitable

Given the growing demand for and impact of interconnect on system cost and performance, it is worthwhile to optimize the utilization of wires. Ad-hoc global wiring struc-

tures often lead to a huge number of wires with an average usage as low as 10% in time [2]. To control cost in this scenario, the wire packing density must be very high, which is not beneficial for the power and delay characteristics. Efficient mechanisms for sharing (semi)-global wires must solve this cost-performance dilemma.

In deep submicron technologies, (semi)-global wires need special attention for power, signal-integrity, and performance reasons. In the discussion below we show how special circuit techniques can handle these issues. Such techniques only work, however, when embedded in dedicated communication IP, which provides a more abstract interface.

Power is an issue for global interconnect because it costs more energy to send a bit of information over longer the wires. To reduce the communication delay, the energy consumption increases due to bigger drivers. Employing low-swing signaling for the global wires saves up to a factor four in power for these wires [15]. Implementing low-swing signaling requires special circuit techniques.

Signal integrity is hampered increasingly by growing capacitive and inductive coupling between wires. Capacitive noise coupling is the result of the large aspect ratio of wires in DSM technologies. Inductive noise coupling becomes more of a problem due to the decreasing transition times. IR drop[1] in the supply distribution increasingly contributes to the noise. The most effective way to make a connection robust against noise is application of differential signaling [7]. Differential signaling improves both the generation of and sensitivity to noise.

The signal propagation delay of an uninterrupted wire grows quadratically with its length; hence from a certain length onwards it is advantageous to partition the wire in segments with repeaters in between. The repeater insertion technique improves bandwidth and latency but at the cost of higher power consumption. Wire delay can be reduced by for wires with a lower resistance per unit length at the cost of lower wire density. Such wires behave like lossy transmission lines and require drivers with a resistance matched to the transmission line.

As a result, we believe that all inter-block communication will be implemented by hard-macro transmitters and receivers, employing low-swing differential signaling, with well-controlled interconnect instead of ad-hoc drivers handled by standard place-and-route tools. In this way, communication links can be realized with predictable performance and DSM robustness.

Currently, the prevalent on-chip interconnects are busses [1]. In a bus architecture, devices share a single transmission medium to communicate. At a given time, only one device has access to the shared medium. An arbitration mechanism is required to order simultaneous accesses. Such functionality is typically performed by a centralized bus arbiter. The performance of a shared-medium bus scales badly. For an increasing number of bus clients (i) individual clients get less bandwidth on average, and (ii) increased capacitive loads and wire length decrease the total bandwidth.

[1] Supply voltage drops are caused by high currents (I) flowing through the resistance (R) of the supply network. Since the supply voltage reduces under scaling IR drop worsens.

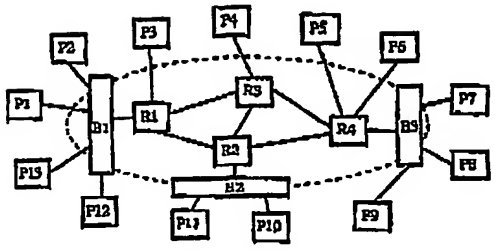

Figure 2, Structural view of a network on silicon consisting of processing nodes (P) and nodes supporting communication (R, B).

A solution that pairs scalable communication performance and minimal interconnect cost is expected from *networks on silicon* (NoS) where the SoS is considered as a network of components [2, 3, 1]. Figure 2 illustrates the hardware architecture of this concept. The outer components (marked P) exclusively perform processing and storage functions, whereas the inner components (marked B and R) form the NoS and cater to communication needs of the outer components. The basic building blocks of a NoS are routers (R).

A router forwards data from its input ports to its output ports in a concurrent fashion. To that end, a router of arity N contains a $N \times N$ switch matrix. Data packets make their way through the network based on the routing information in their headers. A link between two routers is implemented by a point-to-point connection. The links typically span medium to long distances ranging from several to over more than twenty millimeters. The actual length depends on the chosen topology of the network. For a mesh topology the links are relatively short, for a torus which is a mesh with wrap-around connections, some links have a length of half the edge of the chip. Links can be optimized for bandwidth, latency, power, or a combination of these, depending on performance requirements.

3 NoS requirements

An important characteristic of a future system-level architecture is the separation between computation and com-

munication. A NoS allows the computational blocks to communicate with one other via a uniform interface. A uniform interface is advantageous because (i) it frees the core developer from having to make assumptions about the system in which the core will be used, and (ii) does not constrain the development of newer communication architectures by detailed interfacing requirements of particular legacy SoC components [6]. Several on-chip bus standards are evolving to realize this goal, most notably VCI, put forward by VSIA [14], and more recently, the Open Core Protocol [10].

The fundamental aim of a NoS is to provide flexible and efficient communication between the thousands of IP blocks in a system, with performance guarantees. In a typical SoS, the communication demands of different IP blocks show large variations. For example, data rates may be constant (e.g. digital video) or variable (e.g. compressed video). The importance of latency and jitter also varies greatly. Finally, the data granularity may range from single words to large blocks. A NoS should be able to offer different services to different clients. Each service class must be implemented efficiently, using a shared uniform infrastructure.

A high utilization of the network comes at a price. When the network starts to saturate, throughput and latency will show huge variations, which is not acceptable in real-time applications. Hence, the network should also provide guarantees, like loss-less data transport, minimal bandwidth, and bounded latency. The way packets are buffered and scheduled in routers, and the effects on performance guarantees has been the subject of intense research. Fundamentally, sharing and guarantees are conflicting, and efficiently combining guaranteed traffic with best-effort traffic is hard [11]. Although best-effort services are cheaper than guaranteed services we believe that the latter are essential because they enable compositional and scalable integration of the IP blocks [5]. It is up to the IP integrator at design time, and up to the application at run time, to make a trade off.

4 Performance and cost analysis of NoSs

The vision of previous sections is that the design of future SoSs will allow IP blocks to be plugged in at will to minimize communication costs, but without today's problems like timing closure. In this section we investigate the cost implications of system design based on a NoS. We hope the vision comes at acceptable cost. We hope that the overall cost of a NoS, including the full protocol stack to use it, turn out to be acceptable such that the integration blessings of NoSs do not change into a cost nightmare.

4.1 Performance

The aggregate bandwidth of a router is the product of the bandwidth per port, BW_{port}, the arity of the router (number of ports), N, and a utilization factor, $\alpha \leq 1$ corresponding to the router arbitration scheme.

$$BW_{router} = \alpha\, N\, BW_{port} \qquad (1)$$

We discuss each in turn. The bandwidth per port is determined by the bandwidth of the link and the router data path. In short:

$$BW_{port} = B \min(BW_{wire}, BW_{router\_data\_path}) \qquad (2)$$

where B is the width of the data path. The combined bandwidth of the B wires of a link is a function of the layout characteristics (e.g. total length), chosen signaling technique, and the budgets for power, delay, and area. A first-order expression for the bandwidth of a repeated global wire optimized for power-delay is

$$BW_{wire} = \frac{1}{3 \cdot 2 \cdot \frac{2}{3} \cdot FO4} \text{ (bits/sec)} \qquad (3)$$

where FO4 is the delay of an inverter driving four equally sized inverters [4]. In a 100 nm technology, this yields 5 Gb/s per wire under worst-case environmental conditions. Notice that the bandwidth of repeated global wires scales with technology because such wires allow (wave) pipelining at the segments.

Running the router data path at 5 GHz is not feasible. An aggressive but realistic frequency is 1.25 GHz corresponding the clock frequency of 50k gates blocks [4]. The critical function in the data path is the $N \times N$ switch. For N up to 20 it meets the 1.25 GHz data rate, using N 1-out-of-N multiplexors. The relaxed demand on the wires of the link can be used to reduce power dissipation and area.

The utilization factor, α, reflects the effectiveness of the router to resolve contention on the links. The queuing strategy, the queue sizes, and the schedule algorithm all strongly influence α. Accordingly, many queuing policies and scheduling algorithms have been presented in the literature. For example, $\alpha = 0.59$ for infinite fifo input queues with uniform and independent traffic. (Virtual) output queuing gives $\alpha = 1$ under the same conditions, but at the cost of larger queues and a more complex scheduling algorithm [8]. Static scheduling techniques like (time-division-multiplexed) circuit switching can also improve the utilization factor.

Hence, in 100 nm technology, the bandwidth of a 32 bit router port is approximately 5 GByte/sec.

4.2 Cost

Three main components contribute to the area cost of a router: the switch, the control logic, and the packet queues.

The switch allows N simultaneous connections from the N inputs to the N outputs which results in B arrays of $N \times N$ wires, giving rise to an $O(N^2)$ area cost.

The control logic of a router is made up of the switch-matrix schedule unit and other configuration logic. The delay of a schedule cycle varies greatly per algorithm (for example, for virtual output queuing from $O(1)$ to $O(N^{5/2})$ [9]); it is important for two reasons. First, it determines the lower bound for latency that a flit[2] incurs to traverse the router. Second, it affects the size of the queues. The longer a schedule cycle, the more data arrive, given a fixed bandwidth of a port BW_{port}. This leads to deeper queues, and higher area cost.

The three aforementioned queuing strategies require queues of size $O(N)$ to $O(N^2)$ flits. Scheduling algorithms perform better with deeper queues, with a decreasing return.

Besides routers, a significant amount of area is consumed by so-called *network interfaces* (NI) modules. These modules translate the IP transactions for a given connection to packets that are sent over the network, and vice versa. Packets can be sent once the payload has been completely accepted by the NI. Hence, the buffers must be dimensioned such that, at least a complete packet for every simultaneously active connection can be stored.

The trade off between utilization α and the cost is a complex one, but of importance to the viability of NoSs.

5 The future role of busses

In sections 1 and 2 we have argued that NoSs are essential to solve SoS integration in a scalable fashion. While Section 4.2 raised some general cost issues, we will now more concretely consider the trade off between busses and NoSs. *Will packet-switched NoSs completely replace current busses in future SoSs, or will a hybrid approach emerge?* We believe that shared busses may have a role to play in first-level communication (B in Figure 2) for the following reasons.

First, typical IP blocks underutilize the bandwidth capacity of an individual router port. All router ports offer the same bandwidth that is inherent to the architecture, whereas the bandwidth requirements of IP blocks varies greatly. A shared memory module needs typically much higher (peak) bandwidth than a streaming peripheral device. Single word transfers, variable bit rates, bursty IO, and much lower clock rates for IP blocks than for the NoS further waste bandwidth. This means that the communication needs of a number of IP blocks can be aggregated using a bus before the capacity of a network link is reached.

Second, network interfaces are more expensive (in terms of area) than a bus adaptor. Using a bus as a first-level traffic concentrator, trading bus adaptors for network interfaces thus reduces the overall cost of IP-NoS interfacing. We expect that the overhead of a bus and its network interface are outweighed.

[2]Flit stands for flow control digit, the atomic portion of data handled per schedule cycle. A packet is decomposed in flits.

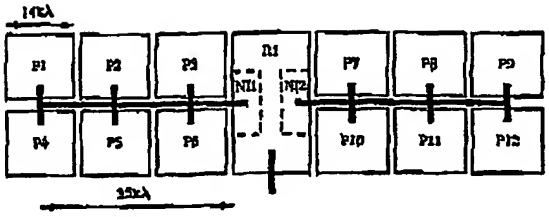

Figure 3. A shared-medium bus seems a cost-effective way to connect the IP to the packet-switched network.

Finally, the number of routers is reduced significantly when busses are used as the first-level interconnect. Routers are larger than busses due to their packet queues and more complex scheduling. We give an example below.

An example of the heterogeneous communication architecture is depicted in Figure 3. A router of arity three surrounded by twelve IP bocks is shown. Two shared-medium busses, each connected to six 50k gates IP blocks, communicate with the router via two network interfaces. These have two functions: first they schedule the transactions on the bus, and second they given the bus clients access to the packet-switched network. The third port of the router provides communication to the remainder of the network. Figure 4 shows an architecture using only routers. Now three routers of arity five and one of arity four are needed.

The suggested shared-medium bus has a length of 35kλ, where λ is half of the length of a minimal transistor. Global wires of this length will not be the bottle-neck of bus performance.[3]

The feasibility of hybrid NoSs hinges on the right implementation of the busses. First, they must be shared wires, as opposed to switches. Second, their arbitration must be combined, or at least compatible with, the scheduling taking place in the network interfaces, to offer uniform end-to-end network services.

We see a future for hybrid NoSs, with first-level communication over a shared-medium bus, and the higher levels using a packet-switched network. Perhaps a packet-switched network can be seen as a distributed and scalable implementation of a logical bridge that connects all the local busses of the SoS. Deciding how many IP blocks can use a local bus

[3]Minimum-delay wire segments have a length of 28kλ, wire segments optimized for power-delay product have a length of 48kλ. These lengths scale with technology like the edge of 50k blocks [4].

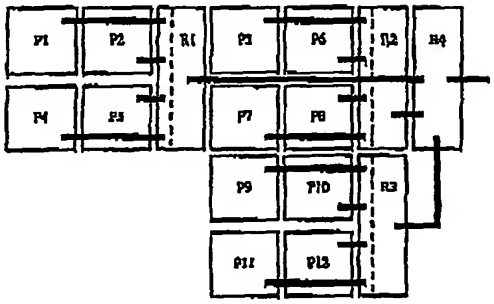

Figure 4. IP to IP communication based on a homogeneous router network.

before connecting to the router network is a question that must be answered foremost.

6 Conclusion

We have argued in Section 1 that future systems on silicon (SoS) will be composed of large numbers of processing nodes (or IP blocks). Each processing node is relatively small (50k gates) to scale with technology, and can be handled by CAD tools, assuming their evolutionary improvement. The interconnect and communication between these blocks then becomes an essential function in itself (Section 2), leading to networks on silicon (NoS). A NoS is based on packet switching to flexibly share link capacity between the network clients, and to provide pluriform communication services over a uniform infrastructure. Both efficiency, provided by best-effort traffic, and predictable performance, such as guaranteed throughput and latency, are important (Section 3). Efficiently combining them is a challenge. Section 4 showed that the performance of a NoS depends on many factors, but is expected to be high. The cost of a NoS can be stated in terms of area (routers, network interfaces), utilization of wires, and speed (latency). They can be traded off against one another, but also, perhaps more interestingly, against the cost of busses. A hybrid NoS using shared-wire busses to communicate locally, and accumulating traffic for a core router network is a promising architecture that deserves to be investigated.

References

[1] L. Benini and G. D. Michell. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, Jan, 2002.

[2] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proc. of DAC*, 2001.

[3] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proc. of DATE*, 2000.

[4] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4), April 2001.

[5] K. Goossens, J. van Meerbergen, A. Peeters and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *Proc. of DATE*, 2002.

[6] K. Lahiri, A. Raghunathan, and S. Dey. Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In *Proc. of Int. Conf. on VLSI Design*, 2001.

[7] Y. Massoud, J. Kawa, D. MacMillen, and J. White. Modeling and analysis of differential signaling for minimizing inductive cross-talk. In *Proc. of DAC*, 2001.

[8] N. McKeown, A. Mekkittikul, V. Ananthsram, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8). August 1999.

[9] N. W. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, University of California, Berkeley, 1995.

[10] Open core protocol specification version 1.0. http://www.sonicsinc.com, 1999.

[11] J. Rexford and K. G. Shin. Support for multiple classes of traffic in multicomputer routers. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 116–130, May 1994. Lecture Notes in Computer Science 853.

[12] S. Rusu. Trends and challenges in vlsi technology scaling towards 100nm (invited paper). In *Proc. of ESSCIRC*, 2001.

[13] D. Sylvester and K. Keutzer. Impact of small process geometries on microarchitectures in systems on a chip. *Proceedings of the IEEE*, 89(4), April 2001.

[14] On chip bus attributes specification 1 OCB 1 1.0, on-chip bus DWG. http://www.vsi.org/library/specs/summary.htm.

[15] H. Zhang, V. George, and J. M. Rabaey. Low-swing on-chip signaling techniques: Effectiveness and robustness. *IEEE Transactions on VLSI*, 8(3), June 2000.

Trade-Offs in the Design of a Router with Combined Guaranteed and Best-Effort Services for Networks on Chip

Edwin Rijpkema, Kees Goossens, Andrei Rădulescu, Jef van Meerbergen, and Paul Wielage
Philips Research Laboratories, Eindhoven, The Netherlands
E-mail: edwin.rijpkema@philips.com

ABSTRACT

Managing the complexity of designing chips containing billions of transistors requires decoupling computation from communication. For the communication, scalable and compositional interconnects (such as networks on chip (NoC)) must be used. In this paper we show that guaranteed services are essential in achieving this decoupling. Guarantees typically come at the cost of inefficient resource utilization. To achieve efficiency, they must be used in combination with best-effort services. We describe a NoC architecture that efficiently combines guaranteed and best-effort services. The key element of our NoC is a router consisting conceptually of two parts: the so-called guaranteed throughput (GT) and best-effort (BE) routers. Both offer data integrity, lossless and in-order data delivery. Additionally, the GT router offers guaranteed throughput and latency services. We combine the GT and BE router architectures efficiently by sharing router resources, enabling high link utilization. The guarantees are never affected by the BE traffic, and links are efficiently utilized because BE traffic uses all bandwidth left over from GT traffic. Connections are programmed using BE packets. The programming model is robust, concurrent, and distributed. It enables run-time and compile-time, deterministic and adaptive connection management. For all our architectural choices, we show the trade offs between hardware complexity and efficiency, and motivate our choices.

1. INTRODUCTION

Recent advances in technology raise the challenge of managing the complexity of designing chips containing billions of transistors. A key ingredient in tackling this challenge is *decoupling the computation from communication* [9, 15]. This decoupling allows IPs (the computation part), and the interconnect (the communication part) to be designed independently from each other.

In this paper, we focus on the communication part. Existing interconnects (e.g., busses) may no longer be feasible for chips with many IPs, because of the diverse and dynamic communication requirements. *Networks on a chip* (NoC) are emerging as an alternative to existing on-chip interconnects because they (a) structure and manage global wires in new deep-submicron technologies [2, 3, 4, 6], (b) share wires, lowering their number and increasing their utilization [4, 6], (c) can be energy efficient and reliable [2], and (d) are scalable when compared to traditional busses [7].

Decoupling the computation from communication requires that services that IPs use to communicate (a) are well-defined, and (b) hide the implementation details of the interconnect [9] , see Figure 1(a). NoCs again help, because they are traditionally designed using layered protocol stacks [14], where each layer provides a well-defined interface which decouples service usage from service implementation [15, 3].

In particular, *guaranteed services* are essential because they make the requirements on the NoC explicit, thus limiting the possible interactions (a stricter contract) of IPs with the communication environment. As a result, IP design is simpler. IPs can also be designed independently, because their guaranteed services are not affected by the interconnect or by other IPs. This is essential for a compositional construction (design and programming) of systems on chip. Moreover, for guaranteed services, failures are restricted to the IP configuration phase (a service request is either granted or denied by the NoC) which simplifies the IP programming mode [6]. We view the guaranteed services to be offered by an interconnect as a requirement from the applications, see Figure 1(b).

The drawback of using guaranteed services is that they require resource reservation for worst-case scenarios. As a consequence, resources may not be efficiently utilized, which may not be acceptable in a system on a chip where cost constraints are typically very tight, see Figure 1(c). To overcome this problem, *best-effort services* can be used for less critical communication requirements to fully utilize the available resources. Using best-effort services, however, provide no guarantees.

A compromise between using guarantees only and having an efficient interconnect is to combine guaranteed and best-effort services. Guaranteed traffic should not be affected by best-effort traffic, while best-effort traffic may use all the resources not used by

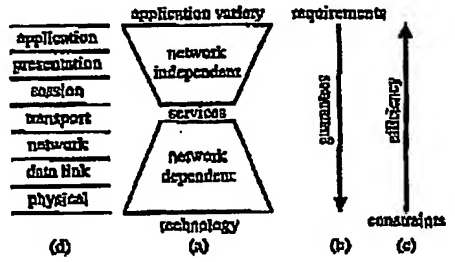

Figure 1: Network services (a) hide the interconnect details and allow reusable components to be build on top of them, (b) are driven by the application requirements, (c) their efficiency relies on technology and network organization, and (d) are build using a layered approach.

guaranteed traffic. Guaranteed services would then be used for the critical traffic requirements, and best-effort services for non-critical traffic requirements.

In this paper, we first list a set of network-independent communication services that are essential in chip design. In the following sections, we show the trade-offs between efficiency and cost that we make in our NoC. In Section 3, we present the trade offs and take decisions on network-related issues. In Section 4, we zoom into the internals of the key component of our NoC: a router which efficiently combines guaranteed and best-effort services.

2. SERVICES

The increasing complexity of integrated circuits, and the strong time-to-market pressure require modular designs and IP reuse. Decoupling computation from communication in chip design serves both these two requirements [9]. This decoupling is realized by defining communication interfaces that provide well-defined *services* and hide the implementation details of the interconnect.

We show in Section 1, that guaranteed services are essential to simplify IP design and integration. Examples of such guaranteed services are *data integrity*, which assumes the data is delivered uncorrupted, *lossless data delivery*, which means no data is dropped in the interconnect, *in-order data delivery*, which specifies that the order in which data is delivered is the same order in which it has been sent. Other guarantees offer time-related bounds, such as *throughput* and *latency*.

Guarantees require resource reservation for worst-case scenarios, which can be expensive. For example, guaranteeing throughput for a stream of data implies reserving bandwidth for its peak throughput, even when its average is much lower. As a consequence, when using guarantees, resources are often underutilized.

Resources are better utilized when best-effort traffic is used. *Best-effort services* do not reserve any resources, and hence provide no guarantees. As a consequence, their performance is dictated by boundary conditions, such as interconnect load. For example, a connection may become temporarily lossy in a congested network, if the network resolves congestion by dropping data.

Best-effort services use resources well because they are typically designed for average-case scenarios as opposed to worst-case scenarios. They are also easy and fast to use, as they require no resource reservation. Their main disadvantage is their unpredictability: one cannot rely on a given performance (i.e., they do not offer guarantees). In the best case, if certain boundary conditions are assumed, a statistical performance can be derived.

The requirements for guaranteed services and the efficiency constraint (good resource utilization) are conflicting. But a first step to a predictable and low-cost interconnect is combining the guaranteed and best-effort services in the same interconnect. Guaranteed services would be used for critical traffic requirements, and best-effort services for non-critical traffic requirements. For example a video processing IP will typically require a lossless, in-order video stream with guaranteed throughput, but possibly allows corrupted samples. Another example is cache updates which require uncorrupted, lossless, low-latency data transfer, but ordering and guaranteed throughput are less important. In Section 4.3 we show how combining guaranteed and best-effort services efficiently uses common resources. In the remainder of this section we analyze the minimum level of abstraction at which the communication services must be offered to hide the network internals.

Traditionally, network services have been implemented and offered using a layered protocol stack, typically aligned to the ISO-OSI reference model [14]. NoCs also take this approach [2, 3, 6, 15], because it structures and decomposes the service implementation, and the protocol stack concepts aid positioning of services.

To achieve the decoupling of computation from communication, the communication services must be offered at least at the level of the transport layer in OSI reference model. It is the first layer that offers end-to-end services, hiding the network details; see Figure 1(d) [3].

The lowest three layers in the protocol stack, namely physical, data-link and network layers, are network specific. Therefore, these services should not be visible to the IPs if decoupling between computation from communication is desired. However, these layers are essential in implementing the services, because constructing guarantees without guarantees at the layer below is either very expensive, or even impossible. For example, implementing a lossless communication on top of a lossy service requires acknowledgment, data retransmission, and filtering duplicated data. This leads to a significant increase in traffic, and also a trade off between large buffer space requirements and long delays. Even worse, providing guarantees for time-related services is impossible if lower layers do not offer these guarantees. For example, throughput can not be guaranteed if communication at a lower layer is lossy. As a consequence, guarantees can only be built *on top* of guarantees, see Figure 1(b). Similarly, a layer's efficiency is based on efficient implementations of the layers below it, see Figure 1(c).

The NoC services that we consider essential for chip design are: data integrity, lossless data delivery, in-order delivery, throughput, and latency. Data integrity is always guaranteed. All the other services can be guaranteed or not, depending on request. In the next section, we describe briefly how these services are provided by our NoC, and in Section 4 we describe in detail how our router architecture enables an efficient implementation of these services.

3. NETWORKS ON CHIP

Currently, the prevalent on-chip interconnects are busses and switches [10]. These are *single-hop* interconnects, meaning that there is no storage in the interconnect itself. Scalable interconnects require multiple hops with storage in every hop (router). This introduces a number of new issues, which we discuss in this section.

General computer network research is a mature research field [16] which has many issues in common with NoCs. However, two significant differences between computer networks and on-chip networks make the trade offs in their design very different [4]. First, routers of a NoC are more resource constrained than those in computer network, in particular in the control complexity and in the amount of memory. Second, communication links of a NoC are relatively shorter than those in computer networks, allowing tight synchronization (network flow control) between routers.

These two characteristics have a direct impact on the NoC service implementation. In a NoC, it is possible to solve the data integrity at the data-link layer at a low cost. We, therefore, assume it solved at the network layer and higher. Lossless transport of data is guaranteed by our routers. However, to allow consumers slower than producers, the network may be allowed to drop data at its edge. Consequently, the designer may choose either for (a) a lossless connection (i.e., implementing end-to-end flow control), or (b) a lossy connection (i.e., without flow control). In-order delivery is again guaranteed by our router (i.e., routers do not reorder data between a given input port and a given output port). End-to-end ordering of data, however, has to be provided on top of this at the network edge when data is transported on different routes with different delays. Offering guaranteed and best-effort throughput and latency services is also implemented by the routers. These router services together with the programming model explained in Section 4.3.2 offer network throughput and latency services.

We identify four important issues in the design of the router network architecture. These are: the *switching mode*, *routing*, *contention resolution*, and *network flow control*. Equally important, *end-to-end flow control* and *congestion control* are handled in our NoC at the network edge instead of the routers; we therefore omit their discussion here.

3.1 Switching Mode

The *switching mode* of a network specifies how data and control are related. We distinguish *circuit switching* and *packet switching*.

In circuit switching data and control are separated. First the control is provided to the network (*connection set up*). This results in a *circuit* over which all subsequent data of the connection is transported. In *time-division switching* bandwidth is shared by time-division multiplexing connections over circuits. Circuit-switched networks inherently offer time-related guaranteed services when resources are reserved during the connection set up.

In *packet switching* data is divided into *packets* and every packet is composed of a control part (the *header*), and a data part (the *payload*). Network routers inspect, and possibly modify, the headers of incoming packets to switch the packet to the appropriate output port. Since in packet switching the packets are self contained, there is no need for a set-up phase to allocate resources. Best-effort services are therefore naturally provided by packet switching.

3.2 Routing

Routing is the determination of the route (or path) that the data follows from source to destination. There are two basic approaches: *source routing* and *destination routing*. In source routing, the network interface at the source computes the complete route to the destination. In destination routing, only the network address of the destination is specified, and every router selects the appropriate output based on the address. We refer to [17] for several classes of routing functions.

In circuit switching, routing takes place at connection set up, i.e., once for all data in that connection. In packet switching, routing is done for every individual packet sent over the network. In both cases, source and destination routing are possible. We currently consider source routing because it is independent of the router network topology, which is not yet determined.

3.3 Contention Resolution

When a router attempts to send multiple data items over the same link at the same time *contention* is said to occur. As only one data item can occupy a link at any point in time a selection among the contending data must be made; this process is called contention resolution. Three approaches exist: avoiding contention, dropping data (one of the contending data item is transmitted and the remainder are deleted), and scheduling (or sequentializing) data (all data items are sent in turn; some data items are therefore delayed).

In circuit switching contention resolution takes place at set up at the granularity of connections, so that data sent over different connections do not conflict. Thus, there is no contention during data transport, and time-related guarantees can be given.

In packet switching contention resolution takes place at the granularity of individual packets. Dropping packets is possible, but for a lossless service (a) it adds complexity to the network (acknowledgments, retransmission, etc.), and (b) it ultimately increases the traffic because dropped packets need to be resent. Thus, scheduling data is the only remaining option.

3.4 Network Flow Control

Network flow control, also called *routing mode* deals with the limited amount of buffering in routers and data acceptance between routers. In circuit switching connections are set up. The data send over these connections is always accepted by the routers and hence no network flow control is needed. In packet switching, data must be buffered at every router before they are sent on. Because routers have a limited amount of buffering they accept data only when they have enough space to store the incoming data.

There are three types of flow control, namely *store and forward*, *virtual cut-through*, and *wormhole* routing. In store-and-forward routing, an input packet is received and stored in its entirety before it is forwarded to the next router. This requires storage for the complete packet, and implies a per-router latency of at least the time required for the router to receive the packet.

In virtual cut-through routing a packet is forwarded as soon as the next router guarantees that the complete packet will be accepted. Only when no guarantee is given, the whole packet is stored in the router. Thus, virtual cut-trough routing requires buffer space for a complete packet, like store and forward routing, but allows lower-latency communication.

In wormhole routing packets are split in so-called *flits* (flow control digits). A flit is passed to the next router when that router accepts that flit, even when there is not enough buffer space for the complete packet. As soon as a flit of a packet is sent over an output port, that output port is reserved for flits of that packet only. When the first flit of a packet is blocked the trailing flits can therefore be spread over multiple routers, blocking the intermediate links. Wormhole routing requires the least buffering (buffer flits instead of packets) and also allows low-latency communication. However, it is more sensitive to deadlock and generally results in lower link utilization than virtual cut-through routing.

We opt for wormhole routing because it offers low latency, which is one of our targeted services, and because it has the lowest cost in terms of buffering, which is expensive on-chip.

4. A COMBINED GT-BE ROUTER

Section 2 defines our requirements for NoCs in terms of services that are to be offered, in particular, both guaranteed and best-effort services. The previous section introduces a number of general networking issues that will be built upon here. In the following two subsections we show that the guaranteed and best-effort services can conceptually be described by two independent router architectures. The combination of these two router architectures is efficient and has a flexible programming model, as described in Subsection 4.3.

4.1 A GT Router Architecture

Our guaranteed-throughput (GT) router must guarantee uncorrupted, lossless and ordered data transfer, and both throughput and latency over a finite time interval. As mentioned earlier, data integrity is solved at the data-link layer; we do not address it further. No data is dropped by the GT router because we use a variant of circuit switching (described in the next section). Data is transported in fixed-size blocks, further explained below. As only one block is stored per input in the GT router, blocks remain ordered. We now turn to the more challenging time-related guarantees, namely throughput and latency.

4.1.1 *Time-related Guarantees*

Latency is defined as the time a packet spends in the network. Guaranteeing latency, therefore, means that a worst-case upper bound must be given for this time. Here we define throughput

for a given producer-consumer pair as the amount of data transported by the network over a finite, fixed time interval. Guaranteeing throughput means giving a lower bound.

We observe that guaranteeing latency in a lossless router is difficult because contention requires scheduling and hence delays. Guaranteeing throughput is less problematic. Rate-based packet switching (for an overview see [18]) offers guaranteed throughput over a finite period, and hence a latency bound. This bound is very high, however, and the cost of buffering is also high. Deadline-based packet switching [13] offers preferential treatment for packets close to their deadline. This allows differential latency guarantees (under certain admissible traffic assumptions), but also at high buffer costs.

Circuit switching solves the contention at set up, so naturally providing guaranteed latency and throughput. Circuits can be pipelined to improve throughput [5], at the cost of additional buffering and latency. Time-division multiplexing connections over pipelined circuits additionally offers flexibility in bandwidth allocation. This requires a notion of router synchronicity, which is possible because a NoC is better controllable than a general network. We explain this variation in more detail in the next subsection. The associated programming model is described in Section 4.3.2.

4.1.2 *Contention-free Routing*

A router uses a *slot table* to (a) avoid contention on a link, (b) divide up bandwidth per link, and (c) switch data to the correct output. Every slot table R has S fixed-size time slots (rows), and N router outputs (columns). There is a logical notion of synchronicity: all routers in the network are in the same slot. In a slot s at most one *block* of data can be read/write per input/output port. The next slot $(s+1)\%S$, the read blocks are written to their appropriate output ports. Blocks thus propagate in a store and forward fashion. The latency a block incurs per router is equal to the duration of a slot. Bandwidth is guaranteed in multiples of block size per S slots.

The entries of the slot table map outputs to inputs for every slot: $R(s,o) = i$. An entry is empty, when there is no reservation for that output in that slot. No contention arises because there is at most one input per output. Sending a single input to multiple outputs (multicast) is possible.

The slots reserved for a block along its path from source to destination increase by one (modulo S). If slot s is reserved in a router, slot $(s+1)\%S$ must be reserved in the next router on the path. The assignment of slots to connections in the network is an optimization problem, and is described in Section 4.3.3. Section 4.3.2 explains how slots are reserved in the network, by means of best-effort packets.

4.2 A BE Router Architecture

Best-effort (BE) traffic can have a better *average* performance than offered by guaranteed services. This depends on boundary conditions, such as network load, that are unpredictable. Best-effort services thus fulfill our efficiency requirement, but without offering time-related guarantees. This section describes an architecture for a best-effort service with uncorrupted, lossless, in-order data transport.

The router efficiency is influenced by both its complexity and its utilization. In Section 3 we have justified our choice for routing (source routing) and network flow control (wormhole). Now we determine the contention resolution scheme that is used. It has two components: buffering and scheduling. Our router prototypes show that the buffering costs dominate the cost of the router. The main trade off in Section 4.2.1 is therefore between buffer costs and link utilization, which are both critical resources. For the chosen buffering strategy an efficient scheduling algorithm is selected in Section 4.2.2, trading off link utilization and schedule complexity.

4.2.1 *Buffering Strategy*

The buffering strategy determines the location of buffers inside the router. We distinguish *input queuing*, *output queuing*, and *virtual output queuing*. In the following, N is the number of inputs (equal to the number of outputs) of the router. We believe that in a balanced solution the rates at which routers and links operate is equal. Slower routers require more buffering, and faster routers are not feasible as links operate at high speed.

In input queuing there is a single queue per input, resulting in the lowest buffer cost (N logical queues in N physical memories) of all three approaches. However, due to the so-called *head-of-line blocking*, for large N network utilization saturates at 59% [8]. Therefore, input queuing results in weak utilization of the links.

Output queuing can increase the link utilization to 100% by having N queues at each output, or N^2 queues, with as many physical memories. It is better to have fewer larger memories than more smaller memories because the overhead of small RAMs is very high. Overclocking the router by a factor N to use N memories is not possible, as argued previously. So the number of memories depends quadratically on N, hence output queuing is not scalable.

Virtual output queuing [1] (VOQ) combines the advantages of input queuing and output queuing. It has the buffering complexity of input queuing and the link utilization of output queuing. As for output queuing, there are N^2 logical queues, but they are combined in N physical memories at the inputs as for input queuing. For every input i there are N queues $Q(i,o)$, one for each output o, see Figure 2. There is at most one write to these queues. The difference between output and VOQ is the additional constraint that there can be at most one read from this group of N queues. (This enabled the mapping of all input queues of the same input to one memory.) This additional constraint has to be taken into account by the scheduling. 100% link utilization can still be achieved, when N is large [12].

We select VOQ because it combines high link utilization with moderate buffer costs.

4.2.2 *Matrix Scheduling*

This section shows how link contention and memory contention (imposed by VOQ) are resolved. Matrix scheduling solves both kinds of contention by ensuring that every VOQ memory is read at most once, and every output (link) is written to at most once. The scheduling problem can be modeled as a bipartite graph matching problem as follows. Every input port i is modeled by a node u_i and every output port o by a node v_o. There is an edge between u_i and v_o if and only if queue $Q(i,o)$ is non-empty. A *match* is a subset of these edges such that every node is incident to at most one edge. For example, Figure 3(c) is a match of Figure 3(a). The number of edges in the match is its *size*; a match is *maximal* when no edges can be added to it. A *maximum size match* is a largest size match.

Although optimal, there are two reasons not to consider only

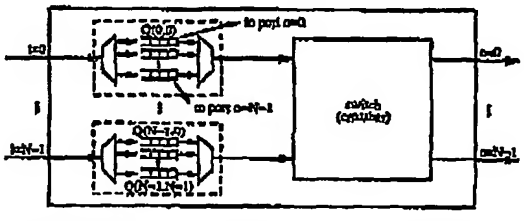

Figure 2: Schematic of a router using virtual output queuing.

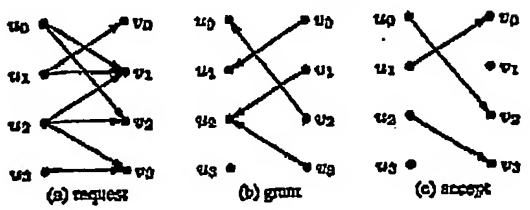

Figure 3: The three steps of a single iSLIP iteration.

maximum size matches. First, maximum size matching algorithms have $O(N^{5/2})$ complexity. Since matrix scheduling is done at flit rate this is not feasible for large N. Second, maximum size matching algorithms can be unfair, which can result in starvation, i.e., some queues are never served.

There are several matching algorithms; see [11] for a thorough discussion. We select the iterative SLIP (iSLIP) matrix scheduling algorithm [11], because it has a low complexity, avoids starvation, and provides increasing performance as the number of iterations grows. It reaches a maximal match in $\log_2(N)$ iterations. Even a single iteration considerably outperforms input queuing, and can be efficiently implemented in hardware. Multiple iterations increase the latency of the control path, and hence the flit size (as explained in Section 4.3.1). We consider using 1-SLIP because multiple iterations give only marginal improvement.

A single iSLIP iteration has three steps, illustrated by an example in Figure 3 for $N = 4$. In the first stage, see Figure 3(a), every non-empty queue $Q(i, o)$ *requests* access to output port o from input port i. In the second stage, see Figure 3(b), every output port o *grants* one request, solving link contention at the output ports. In the third stage, see Figure 3(c), every input port i *accepts* one grant, to resolve memory contention at the input port. We extend iSLIP to take network flow control into account.

4.3 Combining the GT and BE Routers

The GT and BE router architectures are combined to share resources, in particular the links, memories, and switches. Moreover, best-effort traffic enables a packet-based programming model for the guaranteed traffic, as shown later, in Section 4.3.2.

The principal constraint for a combined router architecture is that guaranteed services are never affected by best-effort services. Figure 4(a) shows that, conceptually, the combined router contains both router architectures (fat lines represent data transport, thin lines represent control transport). Incoming data is switched to either the GT or the BE router. The GT traffic (the traffic that is served by the GT router) has the higher priority, to maintain guarantees. This is ensured by the arbitration unit, which therefore affects the best-effort scheduling. Furthermore, best-effort packets can program the guaranteed router, as shown by the arrow labeled program. Thin lines going from the right to the left indicate network flow control, which is only required for best-effort packets because guaranteed blocks never encounter contention.

On a shared link only one BE or GT data item can arrive or be sent at any point in time. Thus GT and BE memories can be shared, keeping the number of memories at N, with $N+N^2$ logical queues in total. Figure 4(b) shows that the data path consisting of memories and switch matrix is shared, and that the control paths of the BE and GT routers are separate, yet interrelated. Moreover, the arbitration unit of Figure 4(a) has been absorbed by the BE router. The following subsection shows how this can be done.

4.3.1 *Arbitration and Flit Size*

When combining GT and BE traffic in a single network the impact on the network flow control scheme must be taken into account. Recall from Section 3.4 that a BE flit is the smallest unit at which flow control is performed. In other words, the BE scheduling, using iSLIP, can only react to GT blocks at flit granularity. To avoid alignment problems, the block size (B words) is a multiple of the flits (F words, with $B = \ell F$). ℓ is constant; we prefer a small ℓ and F to decrease the store-and-forward delay for guaranteed traffic.

We extend iSLIP to handle the combination of GT and BE traffic. In this combination GT traffic always has priority over BE traffic. This is to ensure that guarantees are never corrupted.

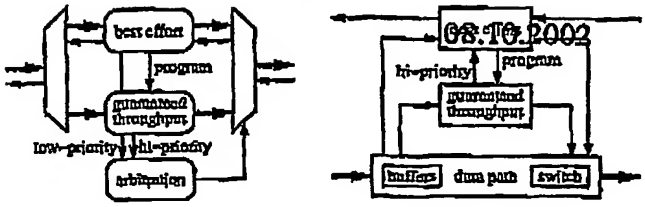

Figure 4: Two views of the combined GT-BE router.

4.3.2 *Programming Model*

In this section we show how GT connections are set up and torn down by means of BE packets. To ensure scalability, programming must not require global or centralized resources. Section 4.1.2 explains why our contention-free routing uses slot tables; we now see that they are distributed over routers for scalability.

Initially the slot table of every router is empty. This means that GT connections can only be set up using BE packets, unless an additional communication infrastructure is introduced solely for programming. Two special packets, Reset and Start, are used to reset and start the NoC, respectively. They progress by flooding, and are not subject to the usual network flow control. We will not discuss them further. There are three *system packets*: SetUp, TearDown, and AckSetUp. They are used to program the slot table in every router on their path.

The SetUp packet is used to create a connection from a source to a destination, and travels in the direction of the data ("downstream"). AckSetUp acknowledges a successful set up, and flows upstream. The TearDown packet destroys (partially) existing connections, and can travel in either direction. SetUp packets contain the source of the data, the path to their destination, and a slot number. Every router along the path of the SetUp packet checks if the output to the next router in the path is free in the slot indicated by the packet. If it is free, the output is reserved in that slot, and the SetUp packet is forwarded with an incremented (modulo S) slot. Otherwise, the SetUp packet is discarded and a TearDown packet returns along the same path. Thus every path must be reversible; this is the only assumption we make about the network topology. These upstream TearDown packets free the slot, and continue with a decremented slot. Downstream TearDown packets work similarly, and remove existing connections. A connection is successfully created when an AckSetUp is received, else a TearDown is received.

The programming model is pipelined and concurrent (multiple system packets can be active in the network simultaneously, also from the same source) and distributed (active in multiple routers). Given the distributed nature of the programming model, ensuring consistency and determinism is crucial. The outcome of programming may depend on the execution order of system packets, but is always consistent. The next section shows how to use the programming model.

4.3.3 *Slot Allocation*

This section explains ways to determine the slots specified in SetUp packets. A slot allocation for a single connection requires that, at every router along the path, the required output is free in the appropriate slot. Therefore, interference of SetUp packets of multiple connections can be completely avoided if connections are set up with conflict-free slots or paths. All execution orders of SetUp packets then give the same result.

Computing an optimal slot allocation is complex and requires a global network view. It can be used only for small problem instances. To reduce computational cost, heuristics can be used, but this probably leads to non-optimal solutions. Compile-time slot allocations from both approaches can be recreated deterministically at run time, concurrently and distributedly (because all SetUp packets are conflict-free).

At run time, a global view requires a centralized slot allocation. This impairs scalability and slows down programming. Run-time distributed slot allocation is scalable, but lacks a global view. This typically results in suboptimal slot allocation. Moreover, SetUp packets may interfere, making programming more involved, and perhaps non-deterministic. However, dynamic connection management at high rates will require distributed slot allocation. In a simple distributed greedy algorithm, all sources repeatedly generate random slot numbers for each set up until their connection succeeds.

We conclude that our programming model allows both compile-time and run-time slot allocation. Computational complexity, deterministic results, and scalability can be balanced according to system requirements.

5. CONCLUSIONS

Managing the complexity of designing chips containing billions of transistors requires decoupling computation from communication. For communication, networks on chip (NoC) are emerging as an alternative for existing interconnects to solve technological, performance, and scalability problems.

In this paper we show that guaranteed services are essential to provide predictable interconnects that enable compositional system design and integration. However, guarantees typically utilize resources inefficiently. Best-effort services overcome this problem but provide no guarantees. So, combining guaranteed and best-effort services allows efficient resource utilization, yet still providing guarantees for critical traffic.

Time-related guarantees, such as throughput and latency, can only be constructed on a NoC that intrinsically has these properties. We therefore define a router-based NoC architecture that combines guaranteed and best-effort services. Thus, the router architecture has conceptually two parts; the guaranteed throughput (GT) and best-effort (BE) routers. Both offer data integrity, lossless data delivery, and in-order data delivery. Additionally, the GT router offers guaranteed throughput and latency services using pipelined circuit switching with time-division multiplexing. This requires a notion of synchronicity: at each time slot at most one block of data is communicated over a link. The GT router has low latency and moderate memory requirements. The BE router uses packet switching, wormhole routing, and virtual output queuing with iSLIP. The BE router has low latency, high link utilization, and moderate memory requirements.

We combine the GT and BE router architectures efficiently by sharing router resources. The guarantees are never affected by the BE traffic, and links are efficiently utilized because BE traffic uses all bandwidth left over by GT traffic. Connections are programmed using BE packets. The programming model is robust, concurrent, and distributed. It enables run-time and compile-time, deterministic and adaptive connection management.

For all our architecture choices, we show the trade offs between hardware complexity and efficiency, and motivate our choices.

In conclusion, we describe and motivate a combined guaranteed and best-effort router, which is an essential component in a NoC. It fulfills our requirements by providing guaranteed services, and satisfies the efficiency constraint by good resource utilization.

6. REFERENCES

[1] M. Ali and M. Youssefi. The performance analysis of an input access scheme in a high-speed packet switch. In *INFOCOM'91*.

[2] L. Benini and G. De Michell. Powering networks on chips. In *ISSS'01*.

[3] L. Benini and G. De Michell. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.

[4] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC'01*.

[5] A. deHon. Robust, high-speed network design for large-scale multiprocessing. Technical Report 1445, MIT, AI Laboratory, Sept. 1993.

[6] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *DATE'02*.

[7] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE'00*.

[8] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Trans. on Communications*, COM-35(12):1347–1356, 1987.

[9] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[10] J. A. Leijten, J. L. van Meerbergen, A. H. Timmer, and J. A. Jess. Prophid, a data-driven multi-processor architecture for high-performance DSP. In *ED&TC'1997*.

[11] N. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, Univ. of California, Berkeley, 1995.

[12] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Trans. on Communications*, 47(8):1260–1272, 1999.

[13] J. Rexford. *Tailoring Router Architectures to Performance Requirements in Cut-Through Networks*. PhD thesis, Univ. of Michigan, 1999.

[14] M. T. Rose. *The Open Book: A Practical Perspective on OSI*. 1990.

[15] M. Sgroi, M. Sheets, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC'01*.

[16] A. S. Tanenbaum. *Computer Networks*. 1996.

[17] A. Varma and C. Raghavendra. *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*. 1994.

[18] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. of the IEEE*, 83(10):1374–96, 1995.

08.10.2002

PH-NL021031

1. Integrated circuit comprising a plurality of modules, and a network arranged for transferring messages between the modules, wherein a message issued by a module comprises first information indicative for a location of an addressed module within the network, and second information indicative for a location within the addressed module,
characterized in that the first and the second information are arranged as a single address from which the network determines which module is addressed, and from which the addressed module determines which of its locations is selected.

2. Method for exchanging messages in an integrated circuit comprising a plurality of modules, the messages between the modules being exchanged via a network, wherein a message issued by a module comprises first information indicative for a location of an addressed module within the network, and second information indicative for a location within the addressed module,
characterized in that the first and the second information are arranged as a single address from which the network determines which module is addressed, and from which the addressed module determines which of its locations is selected.

3. Integrated circuit comprising a plurality of processing modules and a network arranged for providing at least one communication between a first and a second mdule, which communication channel supports transactions comprising outgoing messages from the first module to the second module and return messages from the second module to the first module, characterized in that the network manages the outgoing messages in a way different from the return messages.

4. Method for exchanging messages in an integrated circuit comprising a plurality of modules, the messages between the modules being exchanged via a network, wherein a communication channel through the network supports transactions comprising outgoing messages from the first module to the second module and return messages from the second module to the first module, characterized in that the network manages the outgoing messages in a way different from the return messages.

5. Integrated circuit according to claim 3, wherein the network has a first mode wherein a message is transferred within a guaranteed time interval, and a second mode wherein a message is transferred as fast as possible with the available resources, wherein the outgoing transaction is a read message, requesting the second module to send data to the first module, wherein the return transaction is the data generated by the second module upon this request, and wherein the outgoing transaction is transferred according to the second mode, and the return transaction is transferred according to the first mode.

6. Integrated circuit according to claim 3, wherein the network allows at least two of the following transaction modes unordered, locally ordered and globally ordered, wherein an unordered transaction mode of the network gives no guarantees for the order in which messages will arrive at their destination, a locally ordered transaction mode guarantees that messages sent to the same destination will arrive in the same order as they were sent, a global ordered transaction mode guarantees that messages will arrive in the same order as they were sent even if they are sent to different destinations, wherein outgoing and return transactions are handled according to different transaction modes.

7. Integrated circuit according to claim 3, wherein the network reserves a first and a second buffer space for the first and the second module respectively, the bufferspaces having a mutually different size.

8. Integrated circuit comprising a plurality of modules, which modules are arranged to communicate to each other via a network, wherein the network is arranged to distribute a message from a first module to two or more second modules, and wherein the second modules are arranged to generate an acknowledge message indicating receipt of the message from the first module,
the network being arranged to generate a single return message to the first module, in dependence of the acknowledge messages of the second modules.

9. Integrated circuit according to claim 8, wherein the single return message indicates that at least one of the second modules has received the message issued by the first module.

10. Integrated circuit according to claim 8, wherein the single return message indicates that each of the second modules has received the message issued by the first module.

11. Integrated circuit comprising a first plurality of processing modules and a network, the network comprising a second plurality of nodes and interconnections between nodes, the network being arranged for transferring messages between a first and a second modules via a path through the network, the processing modules coupled to the network via a network interface having a buffer for receiving incoming messages, wherein a message from a first to a second module is not initiated until the buffer has sufficient space for receiving a return message from the second module.

SAMOS, II(), 1–25 (2003)

Communication Services for Networks on Chip

Andrei Rădulescu and Kees Goossens

Philips Research Laboratories, Eindhoven, The Netherlands
Email: andrei.radulescu|kees.goossens@philips.com

Abstract

Networks are emerging as a possible solution for on-chip interconnects. In this paper, we describe how networks on chip (NoC) are similar to and differ from both off-chip networks (e.g., computer networks) and current on-chip interconnects (e.g., buses). We re-examine the communication services in the context of NoCs. We provide services that abstract from network implementations enabling a clean separation between the NoC and IP blocks. We define a request-response transaction model similar to bus protocols, making our approach backward compatible. To exploit the full power of NoCs, we also provide connection-oriented communication with differentiated services. Examples are bandwidth guarantees, transaction orderings, and end-to-end flow control.

Key Words: Networks on chip, on-chip buses, computer networks, communication services, protocol stack, transaction, connection.